Local-to-global in multi-agent systems

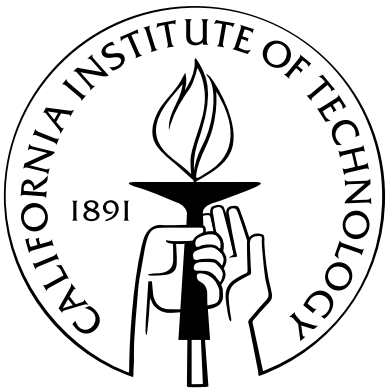
Thesis by

Concetta Pilotto

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science



California Institute of Technology

Pasadena, California

2007

(Submitted May 20, 2007)

To my family

Acknowledgements

I would like to express my gratitude to my advisor, Prof. Mani Chandy, for giving me the wonderful opportunity to work with him. I would also like to thank Prof. Michel Charpentier for useful discussions, suggestions and sharing some of his ideas. I want to thank Agostino Capponi for carefully reviewing the draft of my thesis and providing insightful comments. This research has been supported by the Lee Center for Advanced Networking.

Abstract

The thesis presents performance analysis and simulation results for algorithms that compute global functions out of local interactions on multi-agent systems. We focus on optimization problems; this is because many problems can be formulated in terms of designing algorithms which optimize some global function subject to local constraints.

We model the environment as an adversary of the system. The environment is able to attack the system, modifying the system in arbitrary ways: some agents and/or communication links can be disabled.

Computations proceed by opportunistically employing the resources available at each point, progressing rapidly when more resources are available and slowing down when resources become unavailable.

We investigate and compare two techniques. In the first one, each sub-system (which we call *group*) behaves like a centralized system, i.e., it solves its specific optimization sub-problem by applying a central algorithm. We investigate this technique, called *self-similarity*, showing examples where it works and where it fails, and carry out performance analysis on some problems. Then, we introduce a second technique which is completely decentralized but synchronous: agents simultaneously makes a local update to their current estimates of some true parameter using the estimates of their adjacents. We prove the correctness of this technique on some specific problems by applying tools from distributed systems (variant functions) and control theory (equilibrium points of a dynamical system).

Contents

List of Figures

Chapter 1

Introduction

1.1 Multi-agents systems

A multi-agent system consists of many intelligent, autonomous entities called *agents* interacting with each other in order to perform some task. When all agents share the same final goal, the system consists of *collaborative* agents. When each agent pursues its own interests, we say that the agents of the system are *selfish*. Examples of agents are control systems, software programs or robots.

We focus on collaborative agent systems where data are stored across the system and each agent has only partial capability and/or stores partial information to solve the problem. Agent collaboration is not supervisioned by a global control; therefore, the system is decentralized and agent communication is often asynchronous.

Distributed systems and control theory are useful tools for modeling the behavior of a multi-agent systems and inferring its correctness and soundness. For example, the dynamics of a system can be described by means of differential equations and difference equations, which are standard ways to describe the evolution of a dynamical system in control theory. Correctness and soundness of its behavior can be proved using safety and progress properties which are common tools used in distributed systems for proving correctness and soundness of distributed algorithms.

Throughout the thesis, we assume the communication medium among agents to be wireless, i.e. there is no physical connection between sender and receiver, and there may be possibly noise. We investigate two different communication protocols. In the first chapter we focus on systems where proximate agents establish a bidirectional communication between them, i.e. each agent has a transmitting range and can only communicate with other agents within its range, and only physically close agents can access and modify the local variables of their neighbors. Then, we consider a communication based on identification numbers rather than distance, i.e. each agent corresponds to a unique identifier and stores the identifier of the agents with which it can communicate.

Throughout the thesis, the environment is modeled as an adversary of the system which is able to damage the system by disabling agents and their communication links.

We investigate *homogeneous* systems, i.e. systems where agents have all the same physical and computational capabilities. Agents of the system can be *stationary* or *mobile*. Each stationary agent communicates with a fixed set of agents. Mobile agents are free to move around the area communicating with different agents at different times. This means that communication links between static agents are fixed, while they are mutable when the system consists of mobile agents. We study performance of multi-agent system both in presence of static and mobile agents.

1.2 Agents as distributed systems

In a multi-agent system agents can control only partial portion of the overall area and the data they need to pursue their goal are usually distributed across them.

A distributed system consists of a set of processes which exchange data through a set of channels. The *state* of the system at some time t is defined to be the collection of the states of its processes at time t, which are the set of values of their variables. We denote by $s(t)$ the state of the system at time t and by S the set of all admissible states for the system. Multi-agent systems are distributed systems where the processes are the agents of the systems.

Distributed systems can be modeled using graphs, where vertices correspond to the processes of the systems and edges represent the allowed communications between processes. This representation emphases the communication structure of the system.

Distributed systems can be either *synchronous* (when all processes run using the same clock) or *asynchronous* (if each process has its own independent clock). Synchronous systems are *time-stepped* if processes exchange messages only at pre-specified time interval assuming that the time needed by a process to execute a computation step and send a message to other processes is bounded by fixed value. Asynchronous systems are *shared-memory* if processes of the system share a portion of the memory and use the memory to exchange data or *message-passing* if processes communicate using messages sent and received thorough communication channels.

The state of a distributed system evolves according to some protocol. The desired final state of the system which corresponds to the solution of some problem is called *steady-state* (denoted by s^* throughout the thesis). Hence, starting from some initial state, the goal of the algorithm is to modify the state of the system in such a way that it becomes and remains arbitrarily close to the steady-state forever. *Progress* and *safety* properties are used to formally specify the evolution of the system.

We restrict attention to systems in which progress properties ([6, 8, 16]) assure that, whatever the initial state is, eventually the system remains arbitrarily close to the final state:

$$s(t) \text{ tends-to } s^* \text{ as } t \to \infty : \quad \forall \epsilon > 0 \exists \tau \ \forall t > \tau \Rightarrow D(s(t), s^*) < \epsilon \tag{1.1}$$

where $D(s, s')$ is a measure of the distance between the states s and s'.

Safety properties define the error between the current state $s(t)$ and the desired final state s^*. We consider systems in which a safety property is specified by functions $e : S \rightarrow R$ with the constraint that the function is not nondecreasing with t:

$$t > r \Rightarrow e(s(t)) \leq e(s(r)) \tag{1.2}$$

Hence, combining safety and progress properties, we can prove that the algorithm terminates giving the correct solution.

1.3 Agents as dynamical systems

A thermostat is an example of agent system consisting of a sensor detecting the temperature of some room ([14]). Its goal is to maintain constant the temperature in the room. It pursues its task by turning the heater on if the room temperature is too low and by turning it off when the room is too warm.

This system is a dynamical system, i.e. a system which evolves with time according to some physical laws. Its dynamics can be modeled using differential equations (if agent monitors the environment continuously) or difference equations (if agent monitors at discrete-time steps).

A key concept in studying physical systems is the concept of *equilibrium point* ([2]) of the system which corresponds to a stationary condition (a steady-state) for the dynamics of the system. An equilibrium point is *stable* if perturbing the system nearby the equilibrium point, it remains nearby the equilibrium point; *globally stable* if it is stable for all initial conditions and it is *unstable* otherwise. A powerful tool for determining stability is the use of Lyapunov functions. A Lyapunov function $V : R^n \rightarrow R$ is an non-negative function which decreases along trajectories of the system, making its minimum a locally stable equilibrium point.

The dynamics of a hanging pendulum with a constant string provides some examples for these concepts ([14]). The pendulum hanging straight down is a stable equilibrium position because when its position deviates slightly from the equilibrium, the pendulum always returns to the equilibrium. An example of an unstable equilibrium point for this physical system is the pendulum that points upward. As long as the pendulum points exactly upward, it remains steady. However, when the pendulum deviates slightly from this state, it swings downward and the point leaves the region around the equilibrium value.

1.4 Modeling the environment around the system

Multi-agent systems interact with the environment around them. In the temperature-monitoring agent system example, the environment collaborates with the system by producing the data needed by the agent to performs its task.

Throughout the thesis, we model the environment as an *adversary* of the system. It is able to damage the system: it can disable/enable agents and communication between them arbitrarily. However, it is not able to change the states of the agents or to inject new data in the system.

We assume that environment attacks are randomly distributed across time and space but reversible. Extreme cases where all agents or all communications are permanently disable are not permitted; this is because the resulting system is no longer able to perform the task. Hence, in our model the environment can damage sub-portions of the network both in terms of agents and communication links for arbitrarily long but finite periods of time.

1.5 Modeling multi-agent systems

1.5.1 Internal agent state

Agents are autonomous entities interacting with each others and the environment around them. Agents maintain a *state* consisting of internal data structures where agents record information about the environment and other agents history.

The concept of agent with a state is very similar to the concept objects in object-oriented languages. Objects are computational entities that encapsulate some state, are able to perform actions and communicate with other objects using messages. Hence, agents can be implemented using object-oriented languages. Throughout the thesis, systems are implemented in Java.

1.5.2 Communication graph

Interaction among agents can be formalized by means of a *graph* called *communication graph* where agents are the vertices of the graph and edges are communication links. If there is an edge between two vertices, the corresponding agents can exchange data. From an implementation point of view, communication between agents can be modeled by modifying a shared memory or by exchanging messages.

A compact structure for describing the communication graph is the *vertex adjacency matrix*, which consists of a square matrix whose entries can be either one (if the corresponding vertices share an edge) or zero (otherwise). Each agent is not aware of all the graph structure, but only of its neighborhood. The degree of the agents consists of the size of its neighborhood. The degree of the graph is the maximum degree among the agent degrees.

Agents can communicate using other agents: two agents can exchange data if there exists a continuous path on the communication graph starting at one of them and ending at the other. We can restate the concept of a path in the following way. A path is a sequence of edges of the form $e_1, e_2, \ldots e_k$ such as $e_i \cap e_j \neq \emptyset$. A connected component of a graph is a maximal set of vertices such that there is a path between any two vertices of the set. In a multi-agent system these components are called *groups* or meetings of agents.

Under our model, the graph is not a static structure. The environment attacks can change the structure of the graph by removing vertices and/or edges from the graph. This means that the degree of the agents and the connectivity of the graph can be modified. Agents belonging to the same connected component may be part of different connected components some time later and cannot exchange data anymore.

1.5.3 Global state of the system

The system itself has a state, which defines a global picture of the system and consists of the set of agent states and data currently sent along the communication channels (or stored in shared memories).

Assuming no environmental attacks, the system structure remains constant and the state of the system evolves according to some algorithm: agents run some local programs and modify their states and send messages along channels. When an attack occurs, the system structure changes: some agents can be enabled and other disabled and some communication links may be added and others removed. Hence, the state of the system changes in order to be consistent with the new structure. If environmental attacks are allowed, the state of the system still evolves according to some algorithm except for the times when the state of the system is not predictable due to the incomplete information about the environmental attacks.

1.6 Interdisciplinary of multi-agent systems

1.6.1 Habitat and environmental monitoring: examples from engineering

Classical engineering-based applications of multi-agents systems are distributed systems for habitat and environmental monitoring. Examples of these systems are networks for monitoring the temperature of a building or networks for monitoring vehicles crossing some area. The systems are decentralized and asynchronous; the communication is usually wireless and agents cooperate by exchanging messages. The topology of agents depend on the specific application. In a distributed sensing and tracking system, where each agent needs position and velocity to produce a local prediction for the trajectory of the vehicles, acoustic sensors or radars are usually distributed across

some airspace. Figures 1.1 and 1.2 show possible system architectures for distributed sensing and tracking applications. In environmental monitoring applications, agents need physical quantities such as temperature, pressure, humidity to monitor the environments. Examples of environmental sensors are the US Berkeley Motes communicating with the Mica Weather Board shown in figure 1.3.

Figure 1.1: An example of distributed system architecture where agents are scattered on a 3-dimensional area. Communications among agents are explicitly shown in white

1.6.2 RoboFlag and RoboCup: examples from robotics

Many examples of autonomous multi-agent systems operating in dynamic environments have been designed and developed by the artificial intelligence and robotics community. Using games such as soccer and capture the flag game where two teams play one against the other, researchers are trying to design complex systems capable of producing flexible and dynamic successful strategies: teams must adapt their strategies to score changes and accidents if they want to win the game. Examples of multi-agent systems are the RoboCup and RoboFlag projects. Here, agents are sensors with movement, sensing, tracking and vision capabilities. We refer to the RoboCup web-site (at `www.robocup.org`) and RoboFlag web-site (at `roboflag.mae.cornell.edu`) to get more details about these games.

RoboCup competition has the aim of building a robotic soccer team by 2050. The rules of the

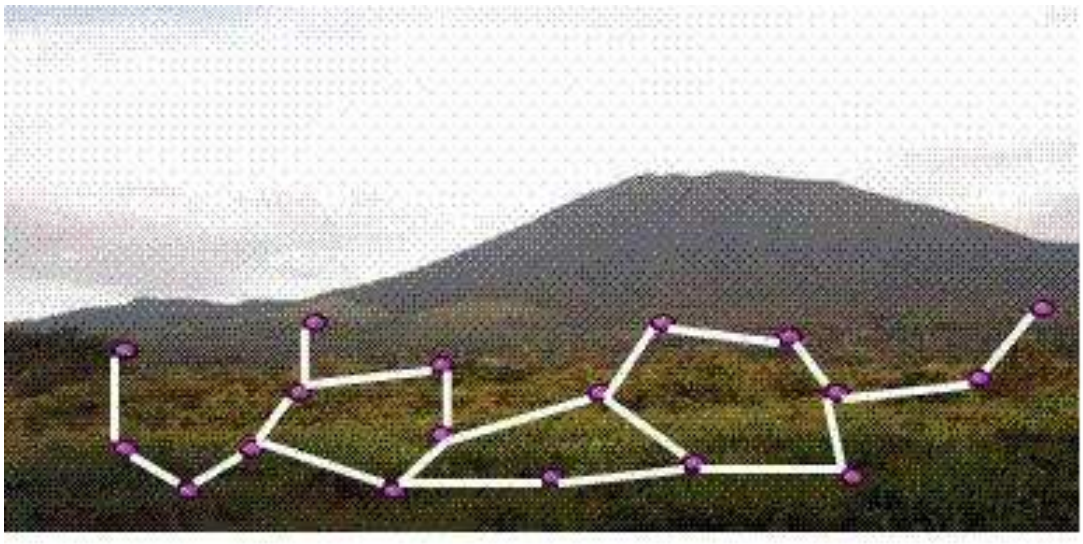


Figure 1.2: An example of distributed system architecture where agents are distributed across a field. The graph explaining the actual structure of the network is shown in white

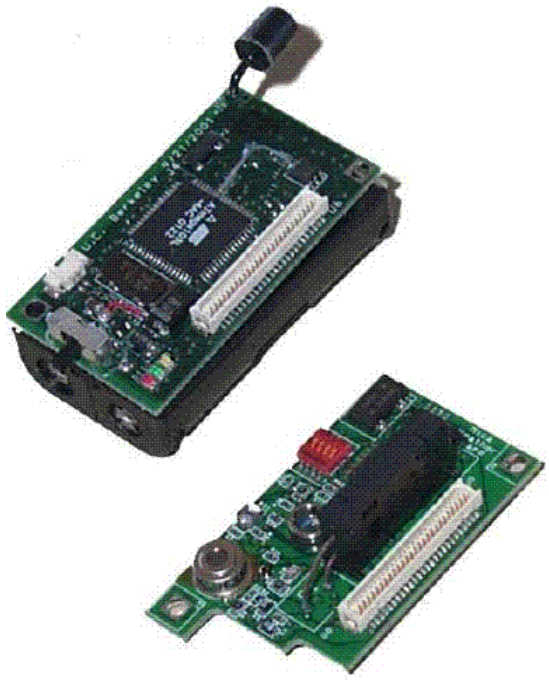


Figure 1.3: The MICA sensor node (top left) with the Mica Weather Board developed for environmental monitoring applications

game are very intuitive: two teams of robots compete against other teams of robots in a soccer match with the objective to score more goals than the opponent.

The RoboFlag game is a successor to the RoboCup competition proposed by D'Andrea and Murray in [11]. Robots play trying to capture the flag game. As displayed in figure 1.4, the game consists of two teams (red and blues) playing against each other. The goal of each team is to locate, capture the opponent flag and return (with the flag) back to the home basis, while defending its own flag.

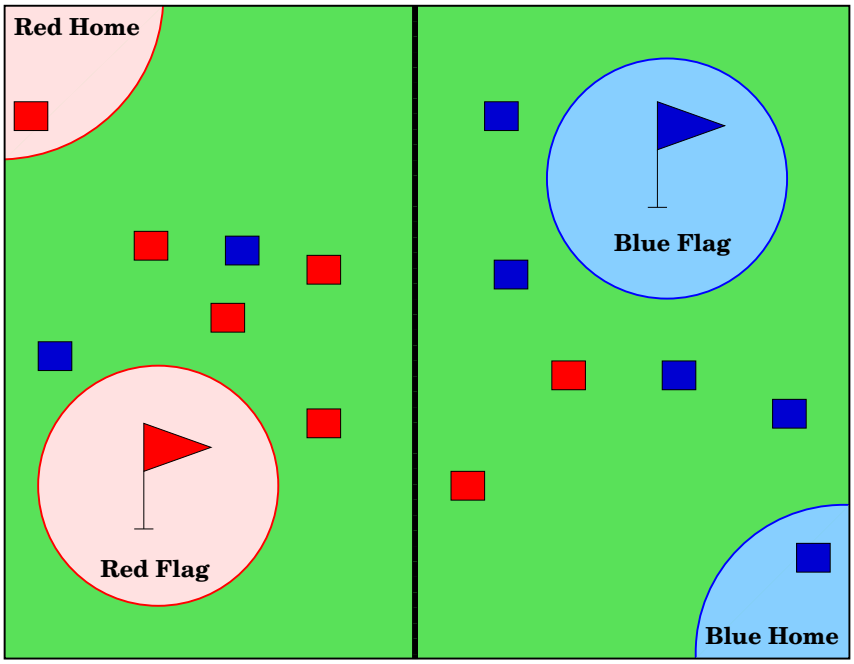


Figure 1.4: The RoboFlag game. Two teams of robots, red and blue, must defend their flag while attempting to capture the other team flag.

1.6.3 Ant colony and flocks of birds: examples from biology

In biological systems, flocks of birds and ant colonies are examples of complex systems, i.e. systems where the group express a complex behavior based on local simple decision. Their collective behavior provides intelligent solutions to daily problems, such as carrying large items, building an ant-hill (where the conformation of hills depend on the amounts of mud and grass in the vicinity), forming bridges and finding the shortest routes from the nest to a food source.

Biological systems have been investigated and simulated using multi-agents systems. For example, the MIT Star-Logo project has designed and implemented multi-agent systems for ant food-hunting and formation of flock of birds.

Figure 1.5 shows some snapshots of ant colony food-hunting system obtained by running the MIT Star-Logo software. For this specific task, ants follow some very simple local rules. As shown

in figure 1.5(A), they start a random walk around the ant-hill. Eventually, one of them discovers a source of food and leaves a trial of pheromone along the path from the food source to the ant-hill (see figure 1.5(B)). While randomly moving ants which find the hormone follow it and reach the food adding some random amount of pheromone to this path (see figure 1.5(C)).

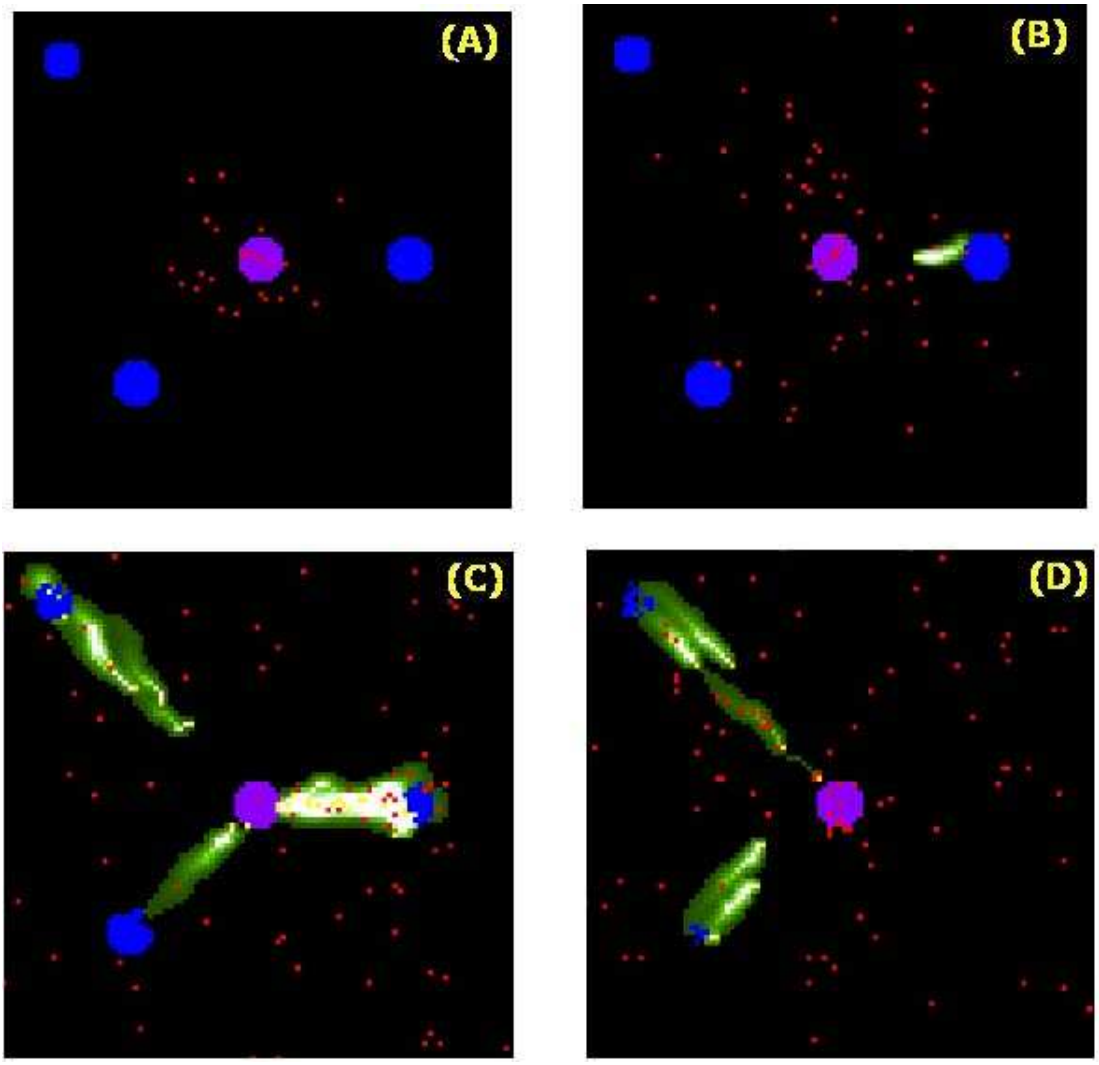


Figure 1.5: Some snapshots from an food hunting ant simulation taken from the Star-Logo project. The violet circle represents the ant-hill while the blue ones correspond to sources of food. A green path is a pheromone trial; ants are represented using red dots.

We refer to the web-site of the Star-Logo project at `education.mit.edu/starlogo/` to get more details about the mechanism of these biological systems.

1.7 Max-min optimization problems

Throughout this thesis, we investigate multi-agent systems whose task is to solve some max-min optimization problem. In this section we detail the structure of the class of max-min problems including some examples. In the class of max-min optimization problem, assuming to have N agents (points) scattered around some field, the state of each agent i includes the following variables:

- V_i, which is the amount of resources allocated to the agent i
- f_i (a function mapping V_i to the reals), where quantity $f_i(V_i)$ is the utility of having V_i resources at location i.

The goal of the agent system is to maximize a system-wide utility z subject to the constraints that resources are conserved and that the amount of resources at each point is non-negative. This can be restated more formally as follows:

$$\text{maximize } z = \min_{1 \leq i \leq N} f_i(V_i)$$

subject to

- **Conservation law**: $\sum_{1 \leq i \leq N} V_i = C$
- **Non-negativity**: $\forall i V_i \geq 0$

where V_i are the unknown (assumed to be scalars) and C is the total initial amount of resource available.

In the next subsections we provide some examples of problems in the max-min optimization class which will be investigated in the thesis.

1.7.1 Consensus problems: average-type problems

Consensus problems can be formulated as max-min optimization problems. For these problems the goal of the system is to agree on some quantity which is a function of the data stored across the system. Consensus problems arise in many practical engineering problems, such as routing and localization problems.

They can be formulated for many order-statistics problems, such as finding the average, the standard deviation of the minimum value (maximum value) across the system. Each of them can be modeled using some specific utility functions. We focus on the average problem.

Informally, for the average problem, each agent stores some real value V_i and the goal of the system is to compute the average of these values in a distributed fashion. A possible instance of the problem (with the corresponding solution) is displayed in figure 1.6; here the structure of the system

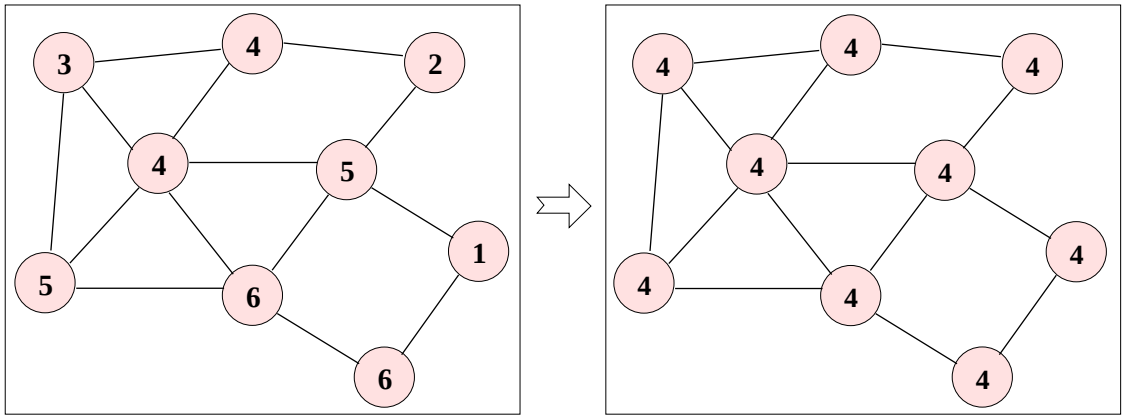


Figure 1.6: An instance of the problem of finding the average of a set of values. The states of the agents are the values stored inside the nodes and the graph shows the communication links.

is visualized using its corresponding communication graph and the state of the agents are included as well storing them in the vertices of the graph.

Agents do not have a global picture of the system; interacting with their neighborhood they receive information about faraway agents. As shown in the left part of figure 1.7, a possible utility function for this problem is the identity function:

$$f_i(V_i) = V_i \tag{1.3}$$

For the problem of finding the average of a set of values, the goal of the system is to maximize the minimum value of these utility functions while maintaining the constant value sum and positivity constraint. Assuming the identity as utility function, it follows that an increase of the values for some of the V_i should be balanced by a decrease of the values for some other. Hence, the maximum is reached when all agents store the same values which is the average of the initial values. We show some pictures 1.7 to better explain the concept. Notice that as long as all agents do not store the average, some of the values can be modified to obtain a better global solution. However, when all agents store the average value, any further change to the values leads to a worse solution.

We report another example of consensus problem which is a generalization of the average problem. This is the problem of finding the weighted average of a set of values and can be formulated as a max-min optimization problem. Each agent i stores a pair of values: V_i (which is the value) and W_i (which is the weight associated to the value V_i). The goal of the system is to compute the weighted average of the values of the agents in a distributed fashion, i.e. finding the value given by the following equation:

$$\frac{\sum_i V_i \cdot W_i}{\sum_i W_i} \tag{1.4}$$

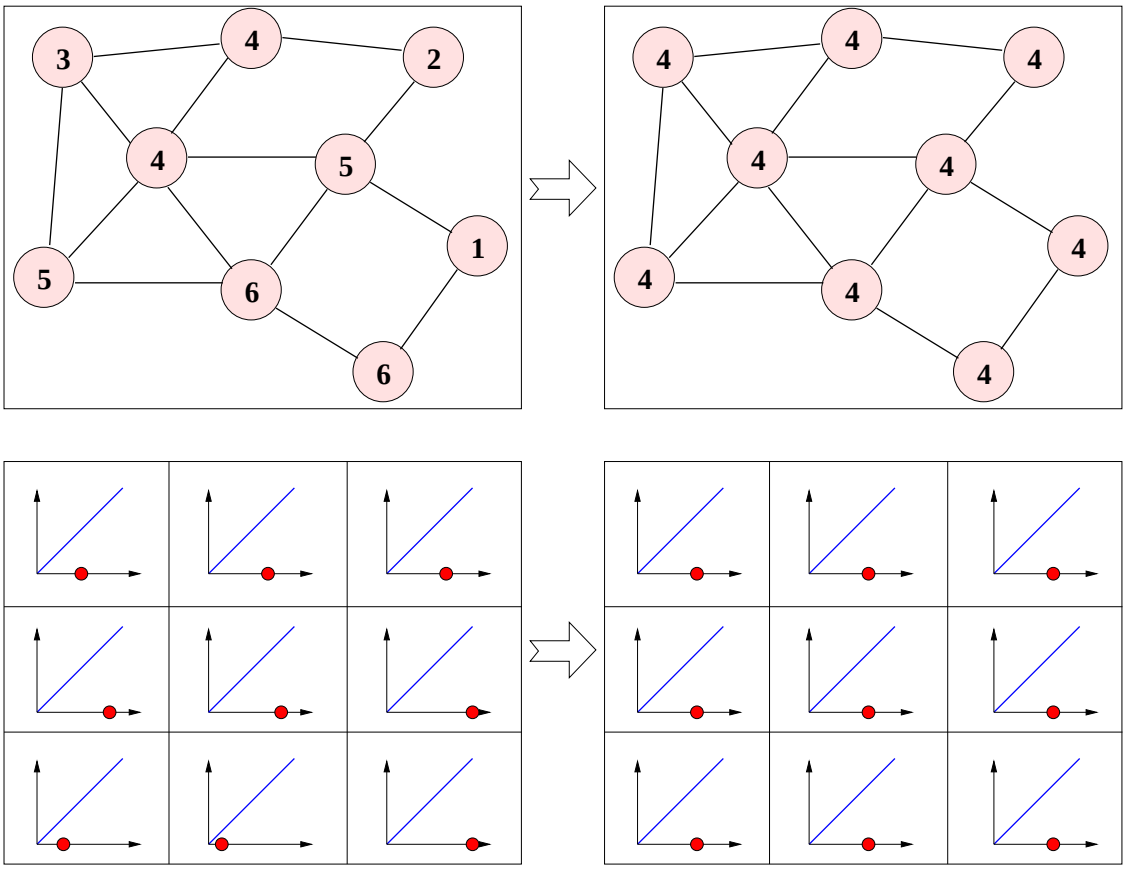


Figure 1.7: The left part of the picture shows an instance of the problem of finding the average of a set of values with the agents utility functions. The right part of the picture shows the solution of the problem with the values of the V_i respect to the utility functions. The red dots on the x-axis of the utility functions represent the current value V_i of the agents.

If we force all weights to be 1, the solution of the equation is the simple average of the values.

To solve the general problem, we can use the following utility function:

$$f(V_i * W_i) = V_i \tag{1.5}$$

where we include the weights in the x-input. This is because for each agent i, the weight W_i can be interpreted as the multiplicity of the agent in the system. The problem of finding the weighted average can be reduced to the simple average problem by replacing each agent i with W_i copies of the agents with value V_i.

1.7.2 Coordination problems: mobile-agent formation

In coordination problems, the system consists of mobile agents whose goal is to move and reach some specific configuration. This class of problems is important for monitoring applications where agents, initially airdropped, should be able to reach pre-established positions without a global coordinator in order to maximize the coverage of the area.

We investigate coordination problems on a line where agents (initially dispersed on the line) try to reach specific configurations described using mathematical equations. We consider the case when agents are not allowed to cross other agents. This means that each agent i is randomly positioned between $i+1$ and $i-1$. Formally, we have $N+1$ agents on a straight line which are indexed from 0 to N. The positions of agents indexed 0 and N are fixed and are the values 0 and C. Each agent stores a value V_i which is the distance between the agent i and its adjacent $i+1$.

For example, agents can try to reach a configuration where they are all equidistant on the line. A possible instance of the problem is shown in figure 1.8, where the values stored in the nodes are the unique identification of the agents rather than the values V_i.

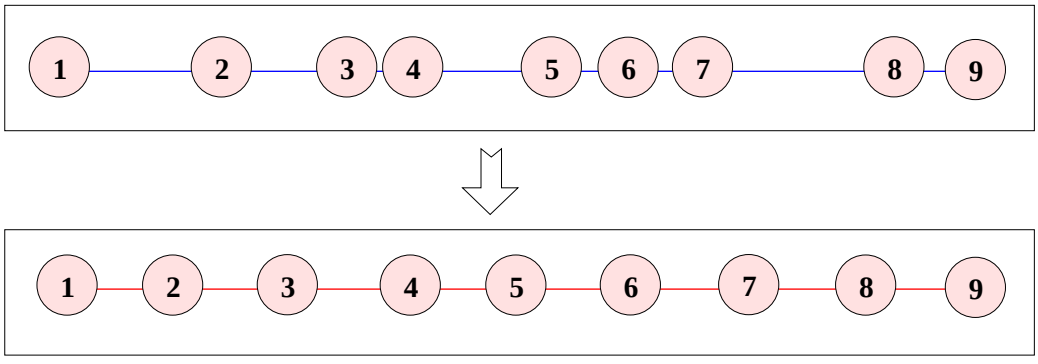


Figure 1.8: An instance of the equidistance problem on the line. Here, agents are randomly distributed between their left and right adjacents and they cannot pass each other. The equidistance configuration is shown in the bottom part of the figure

Agents $i = 1, 2, \ldots N-1$ are required to move along the line so that the V_i's satisfy some criterion;

for the equidistance configuration, we get the following general criterion,

$$V_i = V_{i+1} = \frac{C}{N} = V \tag{1.6}$$

for $i = 0, \ldots, N-1$, where C is the position of the N-th agent. Each agent can retrieve its position P_i by applying the following equation:

$$P_i = \frac{i}{i+1} P_{i+1} \tag{1.7}$$

Assuming no-crossing between agents we get the following constraints on the distances

- $V_i > 0$ for all i
- $\sum_i V_i = C$

In the equidistance problem on a line, agents should converge to the same value V. Hence, the problem can be translated in the problem of finding the average of a set of values summing to C and with utility function

$$f_i(V_i) = V_i \tag{1.8}$$

As a further example, we can require agents to move along the line so that

$$V_i = K V_{i-1} \tag{1.9}$$

for some constant K. In this example, we are not requiring the same distance between any two consecutive agents; instead, the distance increases with agent identification, i.e. expressing all distances in term of the first distance V_0 we have that

$$V_i = K^i V_0 \tag{1.10}$$

A possible instance of the problem with the corresponding solution is given in figure 1.9. Each agent can retrieve its position P_i (with $i = 1, \ldots N-1$) by computing

$$P_i = P_{i-1} + V_i i - 1 \tag{1.11}$$

The problem satisfies the same constraints as the previous problem, i.e. distances are nonnegative and sum up to C. Unlike the previous problem, such a problem cannot be translated to a consensus problem because all distances are different. Hence, the problem can be formulated as a max-min optimization problem with a new utility function

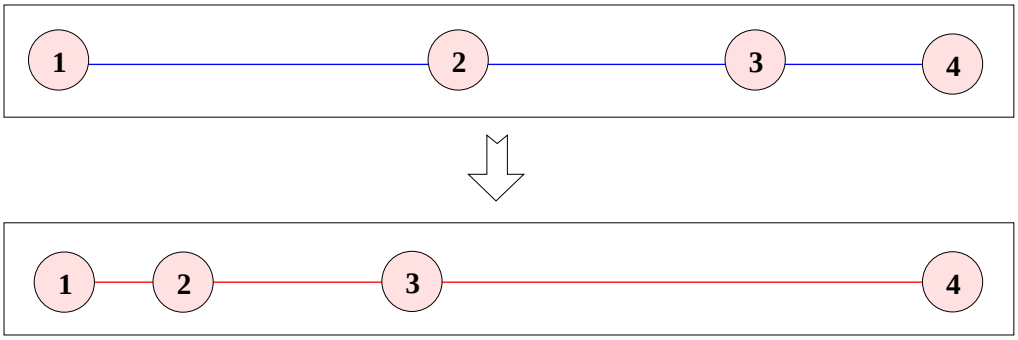


Figure 1.9: An instance of a not equidistance problem on the line. Here, agents are randomly distributed between their left and right adjacents and they cannot pass each other. In this instance the constant K is assumed to be 3. The final configuration is shown in the bottom part of the figure

$$f_i(V_i) = \frac{V_i}{K^i} \tag{1.12}$$

We briefly discuss the case when overlapping is possible for the equidistance problem. An instance of the problem is shown in figure 1.10. If we define the value V_i to be the distance between i and $i+1$ with a positive sign if the position of i is less than position of $i+1$ and negative sign otherwise, we get that while the conservation-law is satisfied, the non-negative constraint is not satisfied. For example in figure 1.10 the distance between 2-3 and 5-6 and 7-8 are negative. Instead, assuming each V_i to be the absolute value of the distance, we get that the conservation law is violated, since some distances are counted several times.

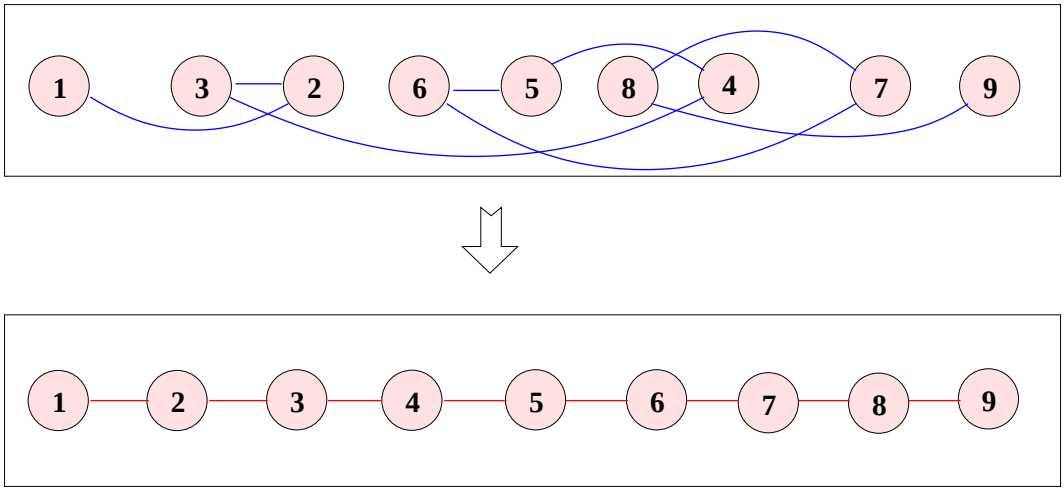


Figure 1.10: An instance of the equidistance problem on the line. The value inside each agents denotes the identification number of the agent.

1.8 Contributions of the thesis

The thesis presents performance analysis and simulation results for algorithms that compute global functions out of local interactions on multi-agent systems. We focus on optimization problems; this is because many problems can be formulated in terms of designing algorithms which optimize some global function subject to local constraints.

In the next four chapters, we model the environment as an adversary of the system. The environment is able to attack the system, modifying it in arbitrary ways: some agents and/or communication links can be disabled. Hence, the environment partitions agents in isolated sub-systems.

We discuss two agent techniques. In the first one (detailed in Chapter 2), sub-systems (which we call *groups*) behave like a centralized system, i.e they solve their specific optimization sub-problem by applying a central algorithm. For example in the average problem, a group can compute the average of its values and each agent of the group modifies its state using the average value of the group. We illustrate this technique, called *self-similarity*, providing examples where it works and where it fails.

In Chapter 3 we carry out performance analysis of the self-similar technique applied to the average problem. We evaluate the impact of group size, operation failure and locality for this problem. We explore some questions about how group formation impacts rates of convergence:

- Should agents tend to form large groups or small groups?
- How does the connectivity of the graph impact convergence?

As our intuition may suggest, algorithms proceeding on large groups converge faster. However, group operations on a large group is more likely to be aborted than an operation on a small group; for example, a component of the group may be disabled while the operation is in progress.

In Chapter 4 we apply this technique on a non-consensus problem where the task of the agent is to build a wall structure with the available resources subject to some constraints. For this problem, we consider a heterogeneous system and deploy two different types of agents: unconstrained agents and constrained ones. The unconstrained agents can build portions of the wall as tall as they can, while constrained agents cannot create wall larger than some threshold. We evaluate the impact of the amount of resources, the threshold on the height of the constrained sub-wall, the number and disposition of constrained agents on the time complexity of the problem.

In the last chapter (Chapter 5), we present another technique already appeared in the paper [24] and we investigate how this technique can be used to solve a generalization of the equidistance problem. For this problem, we assume that agents are allowed to pass each other. We design a synchronous time-stepped algorithm which solve this problem and we prove its correctness using techniques from distributed systems (variant functions) and control theory (equilibrium points of a

dynamical system).

1.9 Literature review

Multi-agent systems have received much attention in the past few years due to the recent developments in the engineering field. Researchers have developed networks consisting of a large number of cheap, customizable, embedded devices capable of wireless communication [42, 1]. These components combine computational, perceptional and control capabilities.

Multi-agents consensus and coordination problems have been widely investigated since they can be used in a variety of applications. For a general overview of the developments and applications in this area we refer to [27, 21]. They have been used in source detection and localization problems [25], routing problems [41, 33], fusion algorithms [43, 19], vehicle formation problems [13], robot synchronization problems [28] and data collecting applications such as weather forecasting, environmental and medical monitoring [3, 5, 23, 26].

Earlier works on decentralized iterative schemes for consensus and coordination problems have been proposed in [12, 9, 36]. Vicsek [37] provides simulation results which demonstrate that local rules can cause all agents to eventually move in the same direction despite the absence of centralized coordination. Theoretical results are given in [29], where they provide a theoretical framework for a network of vehicles with the goal of achieving a specified polynomial formation. They also allow the interaction of the environment which randomly breaks communication links among the vehicles.

In [20, 4, 39, 40, 30, 31] algorithms for distributed computation of averages of the node data over networks are developed. The goal of agents for this type of problem is to converge to the average of their initial states. This is a central problem in signal processing applications [25]. In [34] an iterative algorithm for the average problem is proposed which works if the information topology of the network is fixed, strongly connected and balanced. Although consensus problems are significantly simplified by assuming a fixed network topology, the information exchange topology between agents may change dynamically in real situations. For instance, communication links between agents may be unreliable due to disturbances and/or subject to communication range limitations. In the paper [24] the algorithm is extended to the case when the connectivity of the network varies with time and in [22] an asynchronous version of the algorithm is proposed. In [38] the average problem is investigated for the case when the network is a digraph; this result takes into account that sensors may have a limited field of view which is not always symmetric. Recently stochastic variants of the problems have been investigated in [18] and [35] where the connectivity of the system is assumed to be a stochastic process.

Chapter 2

Dynamic Group Operations

In multi-agent systems, agents store partial information and need to collaborate in order to optimize some global system-wide objective function. In this and next chapters, we investigate meetings of agents as the only possible operation which can be used to make some progress on the problem. During the computation, groups of agents meet and improve locally the current global solution by combining their local information. Their meetings which we call *group meetings* are atomic operations; the current active groups are selected by the environment which disable/enable some agents and/or communication channels. We investigate the convergence of a class of algorithms based on atomic group operations called *self-similar algorithms*.

2.1 Concept of groups

A group is a set of agents cooperating towards some common task where each agent of the group can communicate with and access the states of all other agents in its same group.

Groups are dynamic structures. At each step of the computation some agents may decide to leave and others to join the group. An agent may leave a group because it voluntarily requests to leave, or it is forcibly expelled by other members of the group. Similarly, an agent may join a group; for example, it may have been selected to replace an agent that has recently left the group. Entering and exiting a group is a complex process. Protocols for joining and leaving have been widely investigated in literature ([15], [17], [10], [32]). In our framework, the environment selects groups in an arbitrary manner by disabling/enabling agents and links. We assume to implement some network protocol used by agents to retrieve the members of their own group.

In the communication graph, which models the communication between agents, a group corresponds to a maximally connected component of the graph. These components changes during time, since the communication graph is a dynamic structure under the environment attacks.

In general, the communication among agents in the same group is not immediate, it may be relayed by intermediary agents (belonging to the same meeting). The longer the path between two

agents, the longer the amount of time needed by the two agents to communicate. When agents in a group meet, they perform some operations. The time needed by this operation depends on the structure of the group. For example, if the group is fully-connected, all agents can receive data from the other agents in the group in constant time; instead, if the group is a line, each extreme agent needs a time proportional to the size of the group to get the data from the other extreme. Hence, there is some *propagation delay* when a group operation is executed.

The state of a group is the set of states of the agents in the group. As the computation proceeds, agents change their state according to some algorithm based on the data of the agents in the same group. Group operations are *atomic*. This means that either all agents change their state according to some algorithm or none of them changes. We assume that agents commit their changes by employing protocols that guarantee that changes are atomic.

If, however, there exist some agents which never meet with the others, the computation can be stuck. Throughout the thesis, we assume that certain meetings will take place eventually and that the groups change state eventually (i.e. an operation cannot take a non finite amount of time or fail with probability one). We are implicitly assuming that for every non-empty set s of agents, an infinite number of groups are formed which contains at least one agent from s and one agent from the complement of s.

2.2 Self-similar algorithms

The environment partitions the systems in groups where each group has no information about the existence of the other groups in the system. Each group can solve the problem pertinent to its group using the same optimization steps of the corresponding global, system-wide, centralized algorithm for the problem. This means that each group computes the objective function and constraints pertinent to the group sub-problem and takes steps using the global algorithm that ensure that (1) the local constraints of the agents in the groups are satisfied and (2) the value of the local objective function is improved. These algorithms operate on the state of the group rather then on the state of the entire system; for this reason they are called *self-similar* algorithms because they apply the global algorithm to a local sub-problem.

In order to prove the correctness of these algorithms we need to show that an optimization step executed by any group of agents is also an optimization step of the system, i.e. the following property holds:

Property 1 ***Local optimization steps are global optimization steps.***

In general, the technique can fail, because the property 1 is not satisfied. In [7], the authors define a class of max-min optimization problems for which the self-similar algorithms converge to the true solution of the problem. The main result of their paper follows:

Theorem 1 ([7]) *Self-similar algorithms solve the Max-Min problem if the utility function f_i satisfy the following properties:*

1. *f_i is continuous in $[0, C]$*
2. *$f_i(0) = 0$ and $f_i \geq 0$*
3. *a local maximum of f_i in $[0, C]$ is also a global maximum in the range*
4. *f_i is quasi-concave, i.e. for any points x, y in its domain, the value of the function at any point between x and y is greater or equal to the smaller between $f(x)$ and $f(y)$*

$$\forall x, y \forall \lambda \in [0, 1] \quad f(\lambda x + (1 - \lambda)y) \geq \min(f(x), f(y)) \tag{2.1}$$

We refer to their paper ([7]) for a complete proof of the theorem. In general, acceptable functions may not be monotone, the maximum may not be unique and may not be differentiable everywhere.

Figure 2.1 shows some examples of acceptable and non-acceptable utility functions. The blue functions satisfy the constraints of the theorem 1, while the red functions fail to satisfy some of the constraints. The function on the left is not quasi-concave; if we consider x_1, x_2 as the pair of points we get that the value of the function at z is smaller than the values of the function at x_1, x_2. The function on the right has a local maximum G which is smaller than the global one F.

2.3 Examples of self-similar algorithms

A classical example where self-similar techniques can be successfully applied is the problem of computing the minimum among a set of values. The problem can be stated as follows. Given a set of N agents, where each agent i stores some integer value V_i, the task of the system is to find the global minimum among the values V_i.

Under the environment attacks, groups are formed. When a group meets, the state of each agent i in the group which consists of V_i is modified. The new value for V_i is the minimum value in the group. It is easy to check that the Property 1 holds for the problem. Hence, as the computation proceeds, the system reaches a steady-state consisting of all agents storing the global minimum of the system.

In figure 2.2, we report a sequence of snapshots of the state of the system as agents meets for an instance of the problem.

The utility function for the problems in section 1.7 satisfies the constraints of theorem 1. This means that we can solve them by applying the global, central algorithm to the local group subproblems while converging to the optimal solution. For example, for the problem of finding the average of a set of values, the algorithm becomes: when a group meets, each agent in the group computes

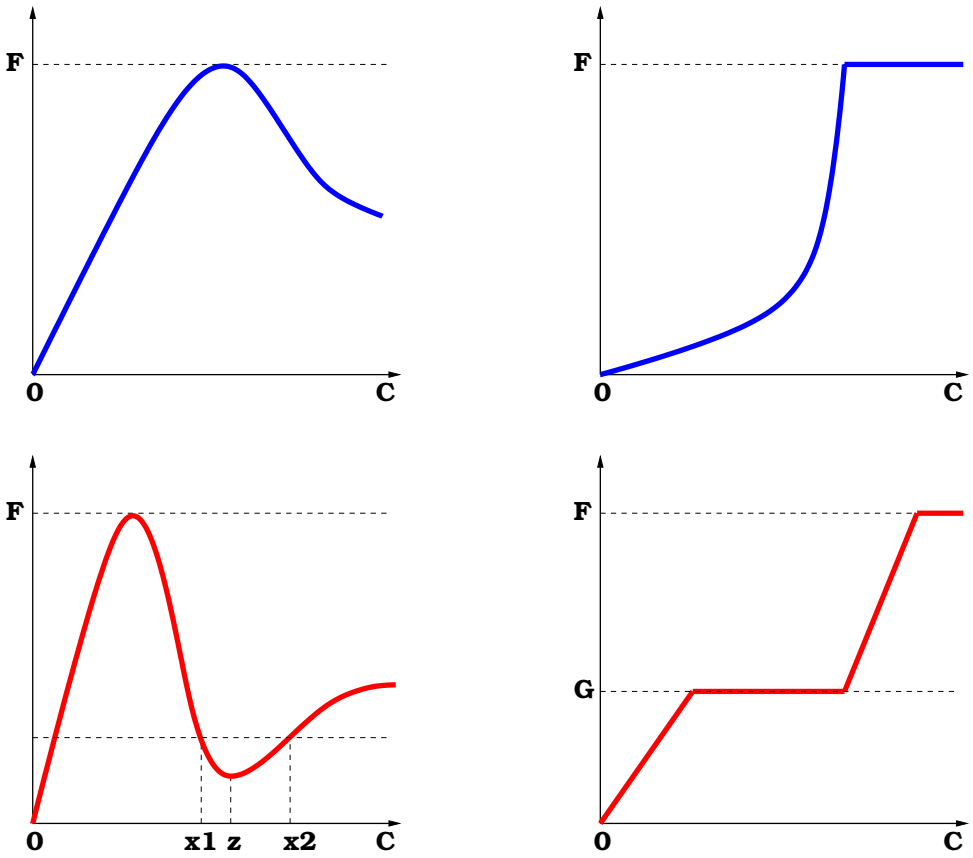


Figure 2.1: Examples of acceptable and non acceptable utility functions. The two function in blue are acceptable, while the red function are not acceptable.

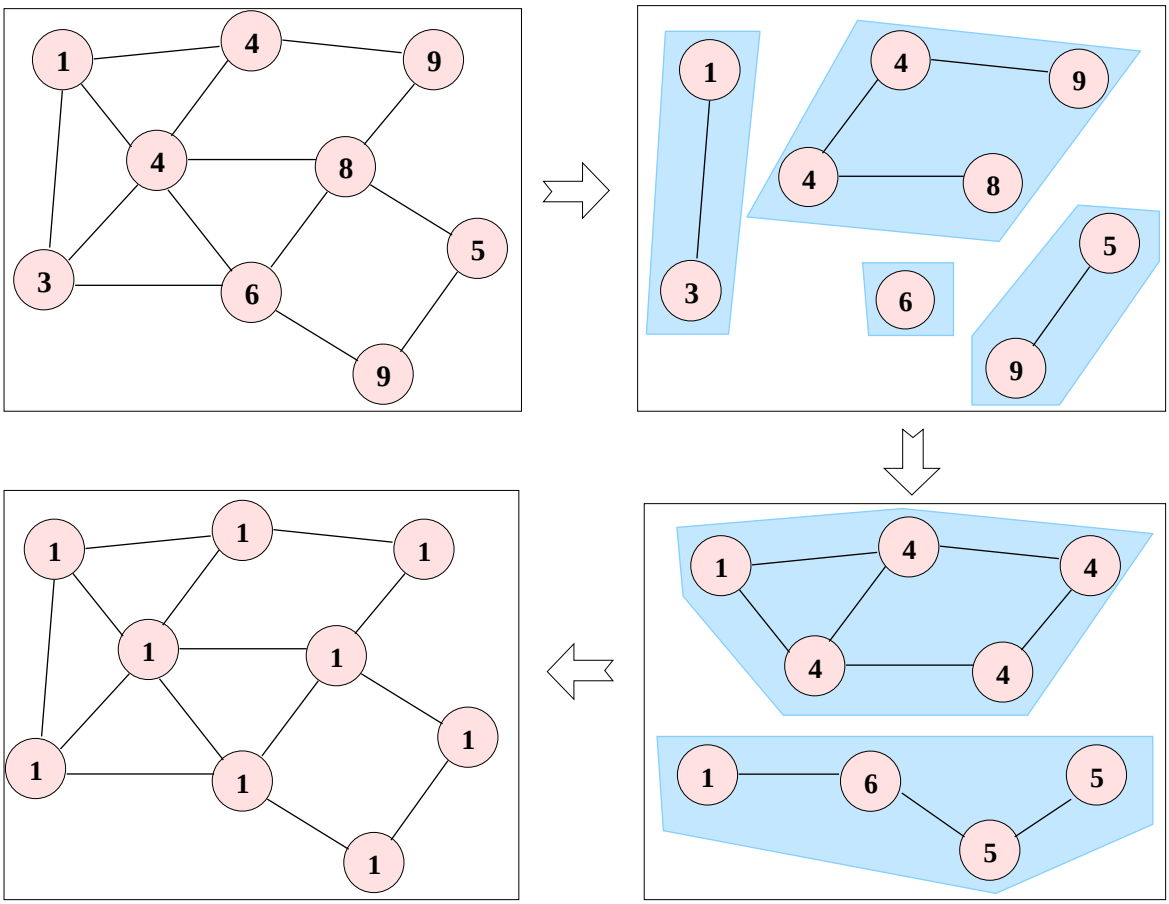


Figure 2.2: A sequence of snapshots of the state of a system for the problem of finding the minimum among a set of values. Numbers inside the agents represent the states of the agents. The geometric blue polygons represent group formation.

the average of the values of the group and replaces its own value with value of the average. In the next chapters, we investigate the behavior of self-similar algorithms for these problems.

2.4 When self-similar technique does not work

Self-similar algorithms cannot solve all problems. We report some examples of problems where self-similar algorithms fail to reach the optimal solution:

- the problem of finding the second smallest value of a system;
- the problem of computing the circumscribing circle of a set of points (i.e. the circle of minimum radius which contains all the given points);

The problem of finding the second smallest values of a set of values can be stated as follows. Given a set of N agents, where each agent i stores some integer value V_i, the task of the system is to find the second smallest values V_i among the given values. A self-similar approach for the problem is very similar to the approach used for computing the minimum of the set of values. When a group meets, all agents of the group find the second smallest value in the group and update their own value with this value. In general, this algorithm does not work; this is because the local-global property 1 can fail to be satisfied, since there are cases where the global second smallest value can be the local group minimum (and hence be discarded). An instance of the problem where the technique fails is shown in figure 2.3. Here, the global second smallest value is 3 while as the computation proceeds the system reaches a steady-state where all agents store the value 8.

We can still solve this problem by running a self-similar algorithm, but we need to solve a more general problem. The new task of the agents in the system is to find both the smallest and the second smallest value of the system. In this problem the amount of resources needed by each agent to store its state increases; this is because the state of the agent i should consist of a pair of values (V_i, W_i) representing the smallest and the second smallest value. When a group meets, both values are updated with the smallest and second smallest values of the group. It is easy to check that the local-global property 1 is satisfied. Eventually, the system reaches a steady-state where all agents store the two smallest values of the system. A sequence of snapshots for an instance of this problem is shown in figure 2.4. Here, agents agree on the same pair of values $(1, 3)$. Hence, 3 is the solution for the problem of finding the second smallest value of the system.

The problem of finding the minimum circumscribing circle around a given set of points can be stated as follows. Given a set of N agents, where each agent i stores a point (X_i, Y_i), the task of the system is to find the circle of minimum radius around there points. Assuming that agents store a pair of values (C_i, R_i) corresponding to the minimum circumscribing circle around the point and all the other points the agent meets, we can use the following self-similar approach for this problem.

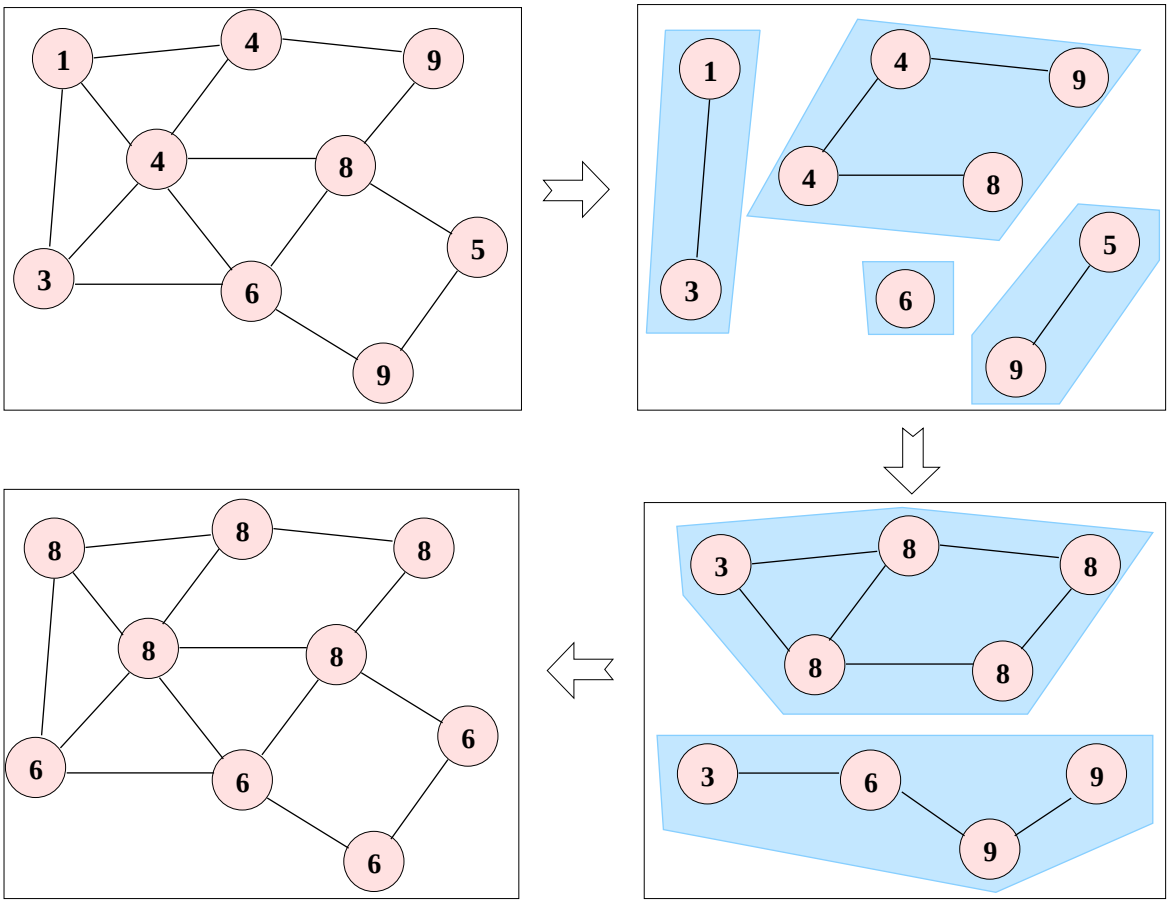


Figure 2.3: A sequence of snapshots of the state of the system for the problem of finding the second smallest value. Values inside the agents correspond to the states of the agents.

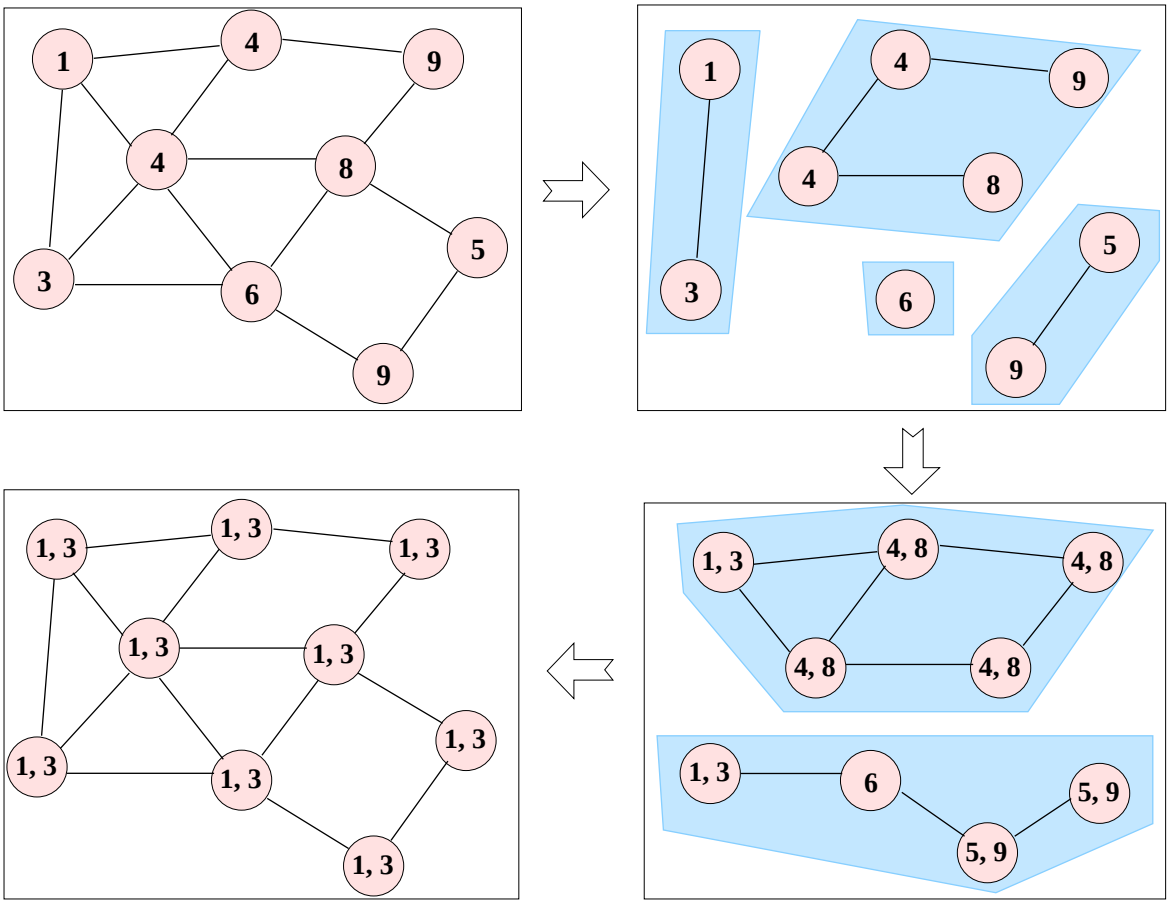


Figure 2.4: A sequence of snapshots for the problem of finding the smallest and the second smallest values in the system. Numbers inside agents represent the state of the agents, where the first number is the value of the variable V_i (the smallest value in the system) and the second is the value of the variable W_i (the second smallest value in the system). Groups are shown in blue.

When a group meets, agents compute the circumscribing circle around the circles of the agents in the group and update their own circle with the new one. Here, the the local-global property 1 fails. Hence, it is not always the case that the system converges to the true optimal solution. An instance where the algorithm fails to solve the problem is shown in figure 2.5. Here, the solution given by the algorithm (blue circle) is much larger of the optimal solution (yellow circle).

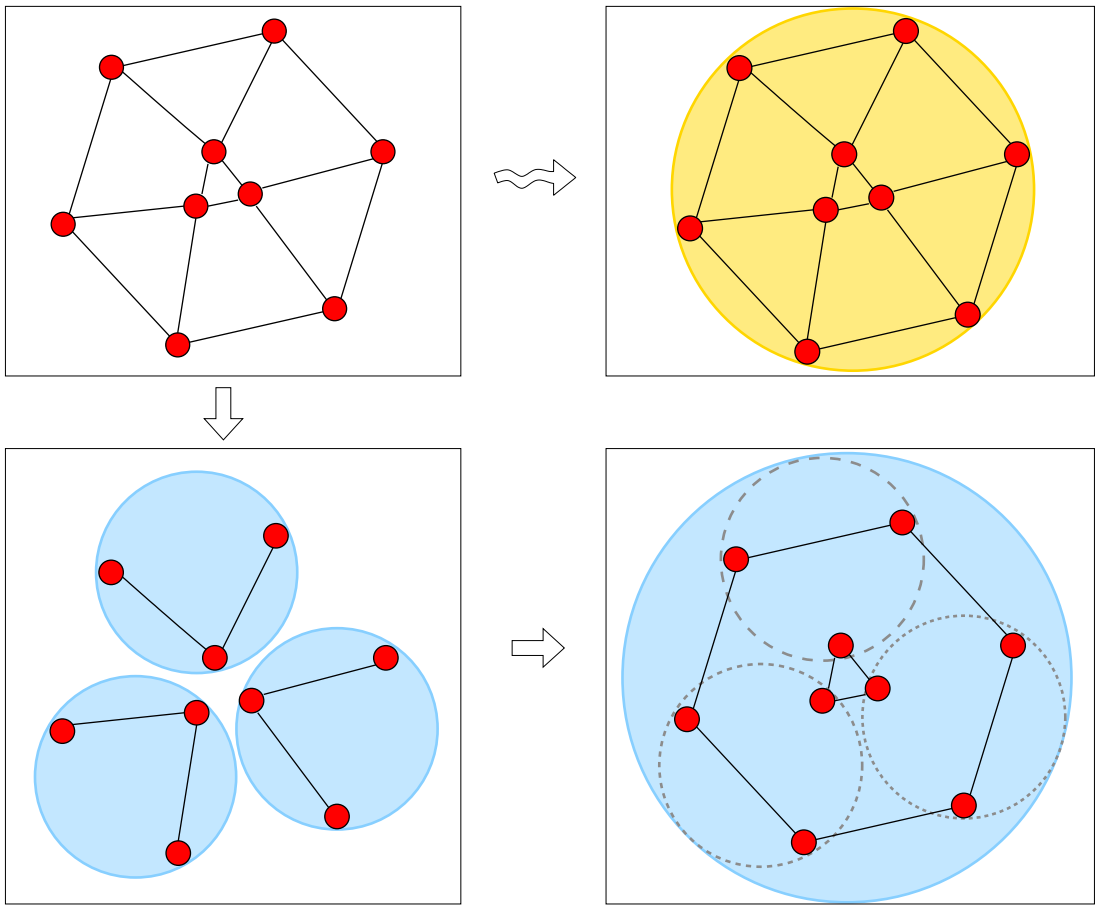


Figure 2.5: A sequence of snapshots for the problem of finding the minimum circumscribing circle of a set of points in the system. The optimal solution is given by the yellow circle while the self-similar algorithm solution is given by the blue circles.

For this problem we can get the optimal solution by solving a similar problem using self-similar techniques. In this case, we can compute the convex-hull of the set of points and, from it, we can derive the minimum circumscribing circle.

Also the problem of computing the linear regression line of a set of points falls in the class of problem where a naive application of the self-similar technique fails.

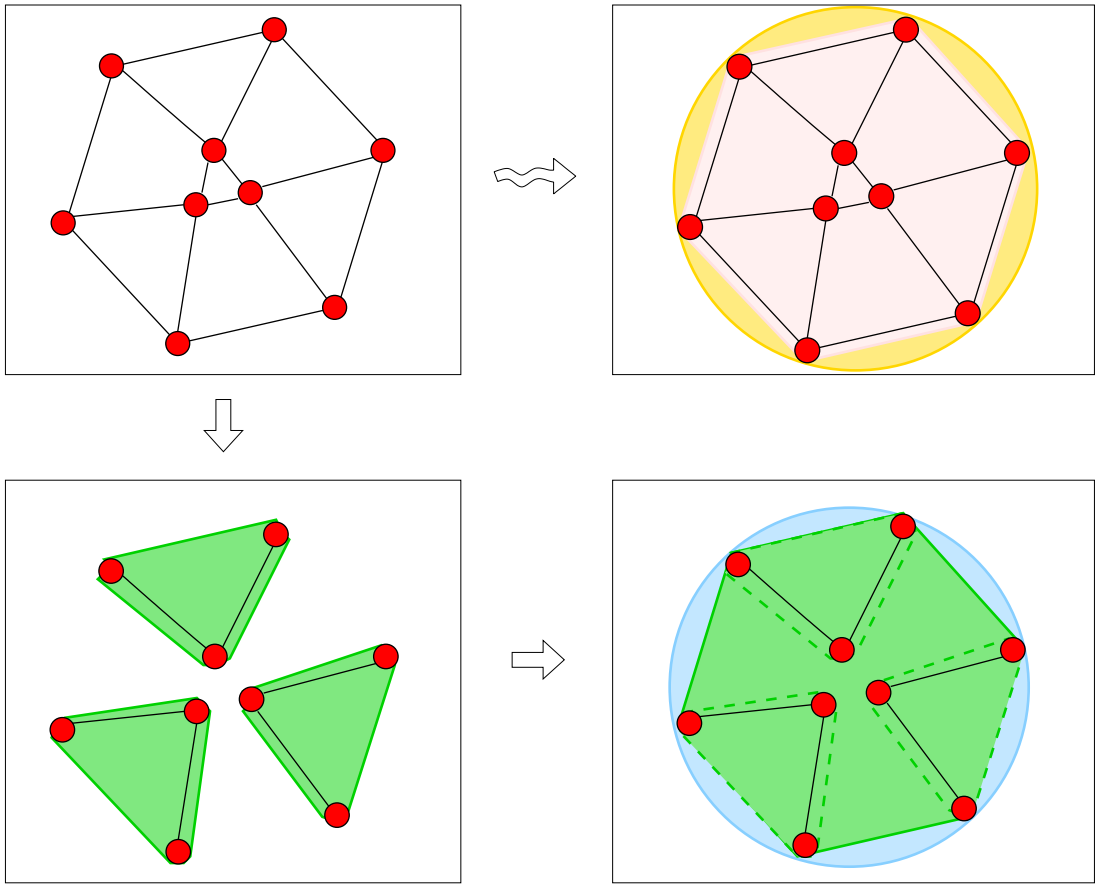


Figure 2.6: A sequence of snapshots for the problem of finding the minimum circumscribing circle of a set of points in the system using the problem of finding the convex hull of these points. Then, using the solution to the convex hull problem, the solution to the problem o finding the minimum circumscribing circle is obtained. The optimal solution for the convex hull problem is given by the pink polygon while the self-similar algorithm solution is given by the green polygon. The optimal solution for the circumscribing circle problem is given by the yellow circle while the self-similar algorithm solution is given by the blue circle.

2.5 Model of failure in group operations

Group operations may fail. This is because the environment can disable agents while they are part of some group. The corresponding group operation should be aborted and the result discarded.

In this thesis, we model failure using probabilities. Denoting by δ the probability that an agent in a group fails during a group operation, we have that the probability that an operation on a group of size n fails becomes the probability that at least one of the agents of the group fails. An explicit expression for this probability can be obtained by rewriting this probability in terms of its complementary event (i.e. the event where all agents do not fail) as follows:

$$\Pr(n) = 1 - (1 - \delta)^n \tag{2.2}$$

In our model, we assume that agents fail independently. In figure 2.7 the probability given by equation 2.2 is plotted as function of the group size for several values of δ. Only when δ is very small we have a positive probability of success for all group sizes. Instead, when δ is larger only small groups have a positive probability of success while larger groups fail with probability $1 - \epsilon$.

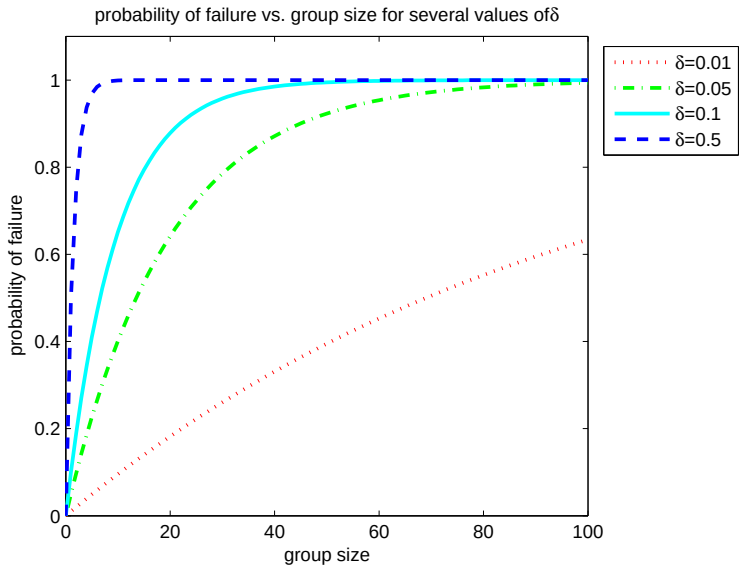


Figure 2.7: Probability of failure versus group size for several values of δ.

Hence, in our model, atomic operations across groups are more likely to abort as the sizes of the groups increases. As a consequence, self-similar algorithms on small groups can result in greater efficiency than the same algorithms on large group sizes because the operations on small groups complete successfully more often than operations on large groups.

2.6 Propagation delay in group operations

When a group meets, all agents in the group collect data of the group, apply the same local optimization step and update their own state with the result of the algorithm. Hence, the time complexity of a group operation does not depend only on the complexity of the local algorithm, but also on the time needed by agents to collect other agent states.

Throughout the thesis, communication links of the system are assumed to be noise free with constant latency. This means that a message sent along a channel is received at the other extreme of the channel without error in a constant (finite) amount of time. Hence, only agents that share directly an edge can communicate in constant time. When agents communicate using intermediary agents, the amount of time needed to transmit some value from the source to the sink is proportional to the length of the path, i.e. there is a physical delay in the communication between two agents which we call propagation delay.

These delays depends on the structure of the group. If the group is a complete subgraph (i.e. any two agents can communicate directly), the delays are negligible, while if the structure of the group is a line, delays are proportional to the size of the group.

We do not assume any special structure for the groups. We illustrate how different structures affect the convergence time of the algorithm. Operations on smaller groups are faster than operations on large group sizes when the propagation delay is proportional to the size of the group. However, large groups can make larger steps towards the solution. We investigate the trade-off between large and small group for several group structures.

Depending on the specific communication model, each agent may require a time which can be linear, polynomial or even exponential in the size of the group to collect agent states.

There is an additional time complexity given by the amount of time needed by agents to form a group (find the members of the group). We assume this time to be negligible.

2.7 Convergence and termination detection

In order to prove that a self-similar algorithm converges to the optimal solution we need to prove the following:

- any group-step maintains global constraints, i.e. local group steps are compatible with the global system steps;
- the algorithm terminates;
- the values at the termination are optimal.

In the paper [7], the authors provide a proof of the correctness of self-similar algorithms for the class of max-min problems with quasi-concave utility functions. The proof is based on providing a variant function h given by the following equation

$$h(G) = sort(f_i(V_i))_{i \in G} \tag{2.3}$$

and check that it eventually increase. In the equation G is the communication graph of the system and the function *sort* provides the vector in non-decreasing order where the total ordering defined on the variant function is given by the lexicographic ordering.

Assuming a continuous state space, we have that self-similar algorithms converge towards the solution, but they never terminate, because they may always make small change to the state.

Following the same approach in [7], we can discretize the state space allowing the values to be modified by some step value Δ. Then we have that the algorithm terminates when no change can improve the current state. Hence, when the algorithm terminates the state of the system is arbitrarily close to the actual solution.

2.8 Implementation issues

We have implemented shared-memory multi-agent systems using the Java language. In our simulations the system is centralized; this means that there is a unique thread of control which runs the simulation and there is a unique central unit which updates agent states. We are modeling agents as passive object modified by a central unit.

There is only one group active in each simulation step. This implementation choice allows us to better analyze the performance of the class of self-similar algorithms. We have not implemented neither a group initialization protocol nor a group coordination protocol; at each step of the simulation, the thread of the simulation forms the current group according to some rules depending on the specific structure of the system and updates concurrently the states of all agents in the groups. We simulate the propagation delay due to the transmission of data between agents in the same group. However, in the simulation we do not take into account for the latency of messages along channels. In a more realistic scenario, each agent of the group should wait to receive the states of all agents in the group before updating its own state. The time spent by agents waiting for messages also depends on latency of communication links.

The environment attacks which decide on the group formation have been simulated using random processes. When the network is fully-connected, agents are chosen randomly according to a uniform distribution to form a group. When the network structure is a line, only the central agent of the group is chosen uniformly among all agents of the system. All other members of the group are on some interval of the line centered in the chosen agent.

The simulation is time-stepped. Several algorithms for detecting termination, which range from local to global termination conditions, have been implemented. While global conditions are easy to implement and detect by a centralized solution, local conditions are more difficult to implement on a centralized systems, but more realistic. As supported by simulation results, we can detect termination faster with global termination condition than using local conditions.

2.9 Performance analysis

We carried out performance analysis of self-similar algorithm for the problem presented in section 1.7. These are based on MonteCarlo simulation with network size varying between 100 and 1000.

We evaluate the impact of group size, operation failure, propagation delay and locality for these problems and draw some conclusions. We explore how group formation impacts rates of convergence: does the algorithm converge faster when operates on large groups or on small groups? As intuition suggests, in a perfect environment algorithms operating on large groups converge faster, because larger groups take bigger steps towards the optimal solution. However, in a more realistic scenario, atomic group operation on a large groups are more likely to be aborted and more time consuming than the same operation on small groups. Hence, it is possible that algorithms that operate on small group size converge faster than algorithm operating with large group size.

Chapter 3

Simulation results for the average problem

This chapter presents performance analysis on the naive self-similar algorithm that finds the average of a set of values. We investigate how group size, operation failure, communication delay and network connectivity impact the performance.

We finally compare our results with the distributed algorithm for the same problem presented in the paper [24], where each agent computes its own estimate based on the estimates of its adjacents.

3.1 Self-similar algorithm for the average problem

The state of each agent consists of a variable V_i which stores the current estimate of the global average and an utility function f_i which we assume to be the line crossing the origin with slope 1 (see figure 1.6.

When a group meets, each agent i computes the average of its value and the values received from all the other agents, and then updates its value with the newly computed average.

Formally, each agent i computes the sum of the received values S and the group utility function g (given by the line crossing the origin with slope equal to $\frac{1}{k}$, where k is the group size) and updates its value with the one given by $f^{-1}(g(S))$.

The utility functions for this problem satisfy the constraints of theorem 1, hence, eventually, all agents in the groups store values which are within a threshold ϵ from the true average.

3.2 Impact of group size on performance

We present data to investigate how group size impacts the performance of the algorithm. Experiments are based on MonteCarlo simulations. The termination condition has been implemented as follows. The algorithm terminates when all estimates are within 0.01 of the true value.

In the first experiment, we investigate the relation between the number of group operations (needed to converge to the true average) and the group size for a fully-connected network. Members of the groups are chosen randomly among all agents according to a uniform distribution. The result of this simulation is shown in figure 3.1 where the x-axis represents the group size varying between 2 and 20, while the y-axis represents the number of group meetings needed to converge in a log-scale. As intuition suggests, the number of time steps of the algorithm drops dramatically with group size, i.e. the larger the group size is, the less the number of operations the algorithm needs to converge.

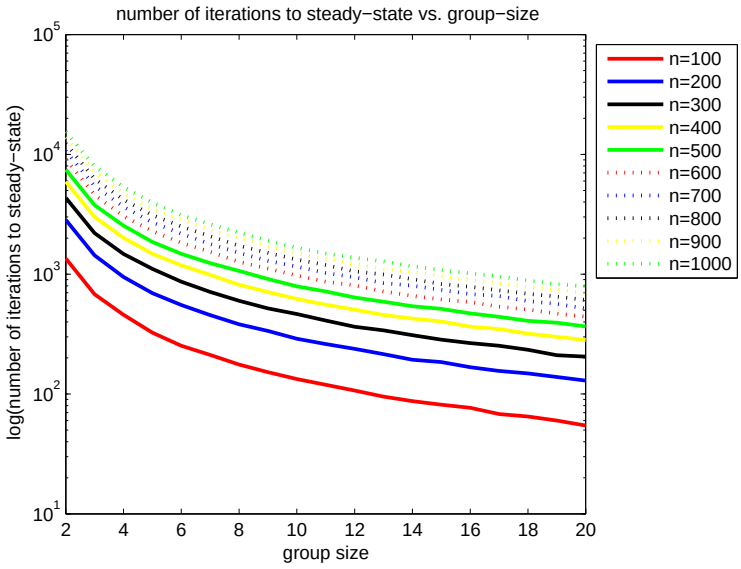


Figure 3.1: Number of group operations to reach the steady-state vs. group size for the problem of finding the average of a set of values. The y-axis is in a logarithmic scale. The network is assumed to be fully-connected. The size of the network n varies between 100 and 1000

As discussed in section 2.6, group operations have some time complexity which we should take into account while analyzing the performance of the algorithm. In general, before applying the optimization step, group membership should be established and data should be transmitted; hence, operations on large groups should be more time consuming than the same operation on small groups.

In the simulation summarized by figure 3.2, we show how the time required by the algorithm to converge varies with the group size, under the assumption that group operations are proportional to the group size. The same is shown for several network sizes. The group size is on the x-axis while the time to convergence is on the y-axis in a logarithmic scale. Differently from the previous figure, the impact of group size is less pronounced; however, we get that the algorithm operating on large group size has better performance than the same algorithm operating on small group size.

This trend changes dramatically when each operation takes an amount of time proportional to the square of the group size ($O(k^2)$, where k is the size of the group). As shown in figure 3.3 we get completely different results: small groups should be preferred to large groups. However, with small

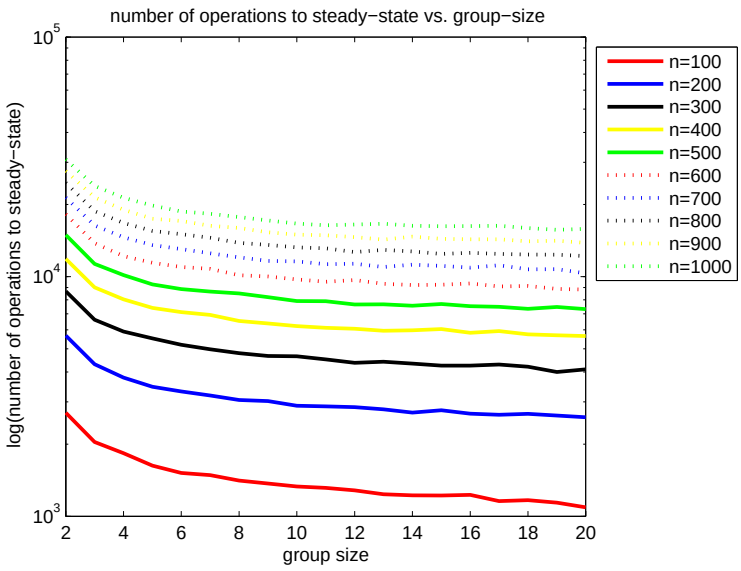


Figure 3.2: Time complexity of the self-similar algorithm versus group size for the problem of finding the average of a set of values assuming group operations to be proportional to group size. The network is assumed to be fully-connected. The size of the network varies between 100 and 1000 and the y axis are on a logarithmic scale.

groups, we need to execute many more group operations than with large group sizes.

We finally present an experiment where the time complexity of the algorithm does not depend on group size, i.e. the algorithm operating on small group (where we execute many cheap operations) is computationally equivalent to the same algorithm running on large group sizes (where we execute few expensive operations). The result of this simulation is presented in figure 3.4.

3.3 Impact of abortion probabilities on performance

In this section, we discuss how group operation failure affects the rate of convergence of the algorithm. We assume that group operations can fail according to the model described in section 2.5 where the probability that an operation on a group of size k fails is:

$$p(k) = 1 - (1 - \delta)^k \tag{3.1}$$

where δ is the probability that a single operation fail.

In figure 3.5, we investigate the relation between the number of group operations (needed to converge to the true average) and the group size when $\delta \in \{0.01, 0.05, 0.1, 0.5\}$. In this experiment, we assume the network to be fully-connected of size 100 (i.e. there are 100 agents). In the figure, on the x-axis there is the group size (varying between 2 and 99), while on the y-axis is reported the number of group operations required to reach the steady-state (in logarithmic scale). The behavior

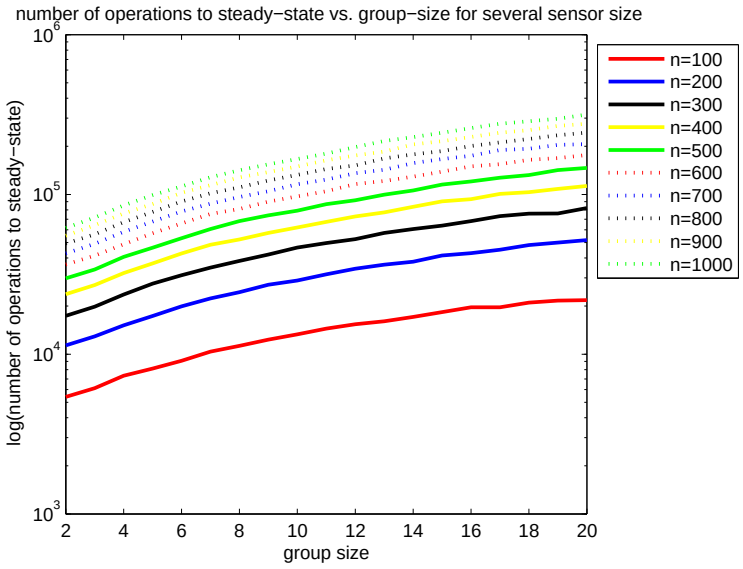


Figure 3.3: Time complexity of the self-similar algorithm versus group size for the problem of finding the average of a set of values assuming group operations to be proportional to the square of the group size. The network is assumed to be fully-connected. The size of the network varies between 100 and 1000 and the y axis are on a logarithmic scale.

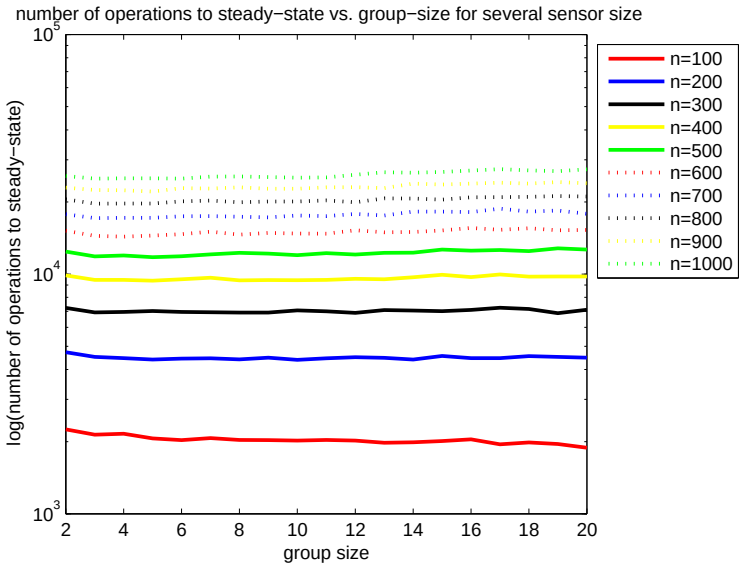


Figure 3.4: Time complexity of the self-similar algorithm versus group size for the problem of finding the average of a set of values assuming group operations to be proportional to $k \log(k)$ where k is the group size. The network is assumed to be fully-connected. The size of the network varies between 100 and 1000 and the y axis are on a logarithmic scale.

of the convergence rate of the algorithm is completely different from the one in figure 3.1. In this experiment, when δ is very large (equal to 0.5), the number of group operations increases dramatically with the group size; hence, we should prefer small group sizes because they complete and commit more often than large group size. When δ is very small (0.01), the number of operation decreases slowly with the group size, making us preferring large group sizes. When δ is in the interval between 0.05 and 0.1, we get that the number of operations is a parabolic function of the group size, it first decreases and then increase reaching its minimum at a group size equals to 10 when $\delta = 0.05$ and equals to 25 when $\delta = 0.1$. When δ is smaller than 0.01 the algorithm converges for all group sizes. However, as δ increases to 0.05, only when the group size is less than 20 the algorithm converges; any further increase in group size is deleterious to the algorithm since the probability of failure for these group size becomes almost 1 under our model (see figure 2.7).

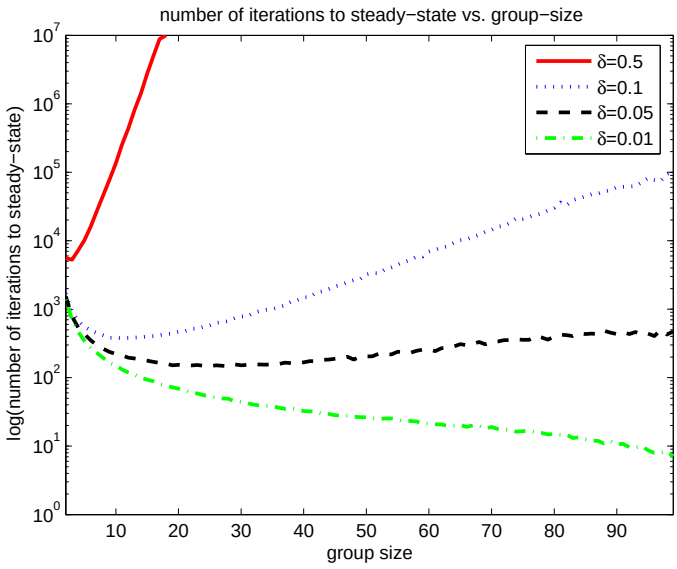


Figure 3.5: Number of group operations vs. group size for the average problem when group operation can fail according to equation 2.2. The network consists of 100 agents and it is fully-connected. The group size varies between 2 and 99, the number of operations varies between 10 and 10^7. The values on the y-axis are in a logarithmic scale. The values of δ ranges between 0.5 and 0.01

In practice, group operations have some non constant time complexity which we should take into account when analyzing the performances of the algorithm. In the following experiments, we assume that group failures have the same time complexity as the group operations. This is a reasonable assumption since all agents in the group should be informed to discard the result of the operation when the operation fails.

In figure 3.6, we show the simulation when both the time complexity of group operations and group failure is linear in the size of the group. The new results are not very different from the graphs presented in the previous figure. When δ is very small, large groups should be preferred, while when

δ is very large, small groups should be preferred. Finally when δ is between 0.05 and 0.1, the best group size is obtained at 3 (if $\delta = 0.1$) at 7 (if $\delta = 0.05$).

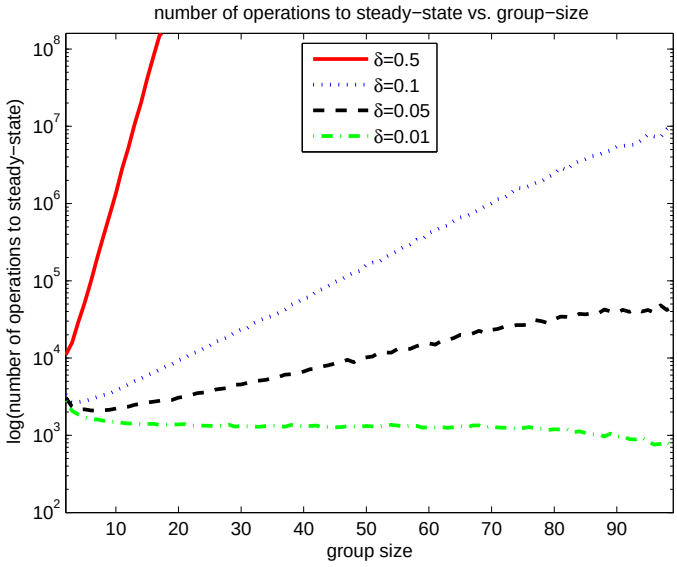


Figure 3.6: Number of group operations vs. group size for the average problem when group operation can fail according to equation 2.2 and the time complexity of group operation is proportional to group size. The network consists of 100 agents and it is fully-connected. The group size varies between 2 and 99 while the number of operation varies between 10^2 and 10^8. The values on the y-axis are in a logarithmic scale. The values of δ ranges between 0.5 and 0.01

In figure 3.7, we present the results for the case when the time complexity of group operations is proportional to k^2. The results are very different from the one presented in figure 3.5. When δ is larger that 0.05, the time complexity of the algorithm increases with the group size, making us prefer small group sizes. When $\delta = 0.01$, the time complexity function has a bell shape, it first increases and then decreases, making us preferring extreme cases (very small or very large group size).

3.4 Impact of group locality on performance

All experiments discussed so far refer to a fully-connected network. In this section we investigate how performances changes when the network is a line.

While in a completely-connected graph groups of agents are picked uniformly from the set of agents, for the linear graph groups are linear segments on the line. We executed two sets of simulations. In the first one the segments are picked uniformly at random at each step. In the second one, the group window of size k right shifts at each iteration of one position. For example, if the group size is 2, then after a group of agents indexed (0, 1) is picked, a group consisting of agents (1, 2) is picked and so on.

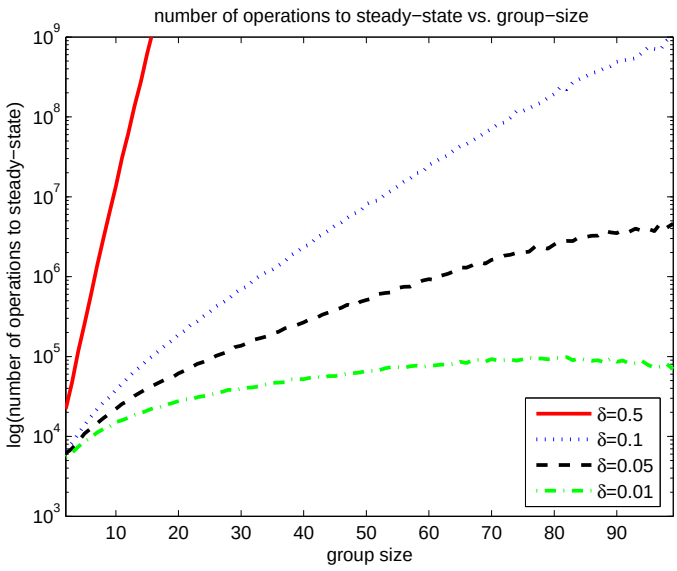


Figure 3.7: Number of group operations vs. group size for the average problem when group operation can fail according to equation 2.2 and the time complexity of group operation is proportional to square of the group size. The network consists of 100 agents and it is fully-connected. The group size varies between 2 and 99 while the number of operation varies between 10^2 and 10^8. The values on the y-axis are in a logarithmic scale. The values of δ ranges between 0.5 and 0.01

In figure 3.8, the number of operation needed to reach convergence as a function of group size is shown when the network is fully connected and when it is a line. The network consists of 100 agents and the group size varies between 2 and 20. On the y-axis the number of iterations is presented in logarithmic scale. The number of iterations is dramatically lower for completely connected graphs than for sparse graph. Considering the linear graph, we get that the deterministic algorithm is better than the probabilistic one for small group sizes but worse for large group sizes. A possible explanation for this phenomenon follows.

For the linear graph, we assume that agent i is connected to agents $i-1$ and $i+1$, and consider the case where agent i has value i, for $1 \leq i \leq 10$. Consider the sliding window algorithm with a window size of 2. It sets the value of agents 0, 1 to 0.5 each; then it changes values of agents 1 and 2 from 0.5 and 2 to 1.25 each; then agents 2 and 3 change their values from 1.25 and 3 to 2.125 each, and so on. At each succeeding step the values of agents in the group will differ by more than 1. Consider the random algorithm that picks pairs say 1, 2 and then 8, 9 and then 5, 6; in each of these steps the values in each group differ by 1. Thus, at least initially, the value of the Lyapunov function h decreases more rapidly for the sliding window case. If, by contrast, the number of elements is large, say 1000, and the window size is also large, say 100, then sliding the window by 1 at each step reduces h by less, on the average, than picking a random window of the same size.

In figure 3.6, we show the behavior of the simulation when the time complexity of a group

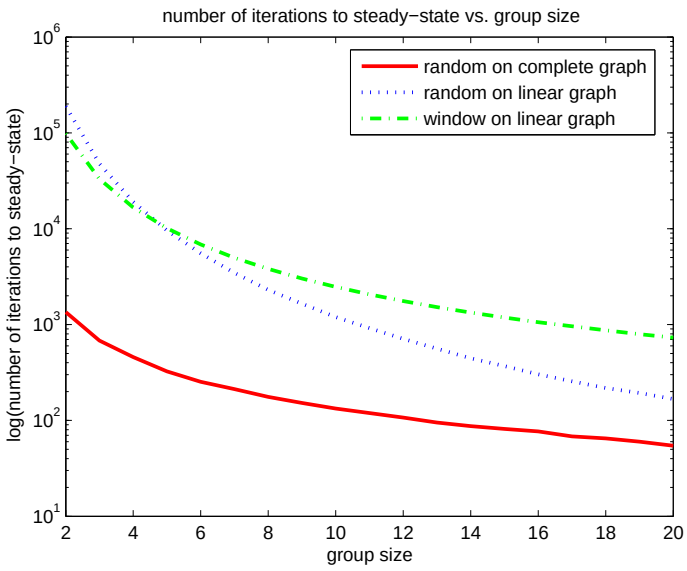


Figure 3.8: Number of group operations versus group size for several network for the problem of computing the average of a set of values. We compare the time complexity of the algorithm on a fully connected network with its complexity on a line. When the network is fully-connected, group members are chosen randomly. When agents are on the line we report simulation results for two different group members heuristics. The group size varies between 2 and 20. The y-axis is on a logarithmic scale.

operation is linear in the size of the group. The new results are very similar to the previous one; even if, it is worthwhile noticing that the gap between the fully connected case and the linear case increases dramatically.

As shown in figure 3.3 we get completely different results for the fully-connected case when the time complexity of a group operation is proportional to the square of the size of a group. In this case, small groups should be preferred to large groups. In figure 3.10, we investigate whether the linear case has the same behavior. When the group are chosen according to the deterministic heuristic we get that the time complexity is a parabolic function; it first decreases and then increases, making us preferring groups of size 5. For the random heuristic, we get that the time complexity decreases as group size increases. However, the gap between the fully-connected case and the linear cases is smaller than the one presented in the previous picture.

3.5 Comparison with an algorithm for synchronous time-stepped network

We compare the performance of this self-similar algorithm with the algorithm presented in the paper [24].

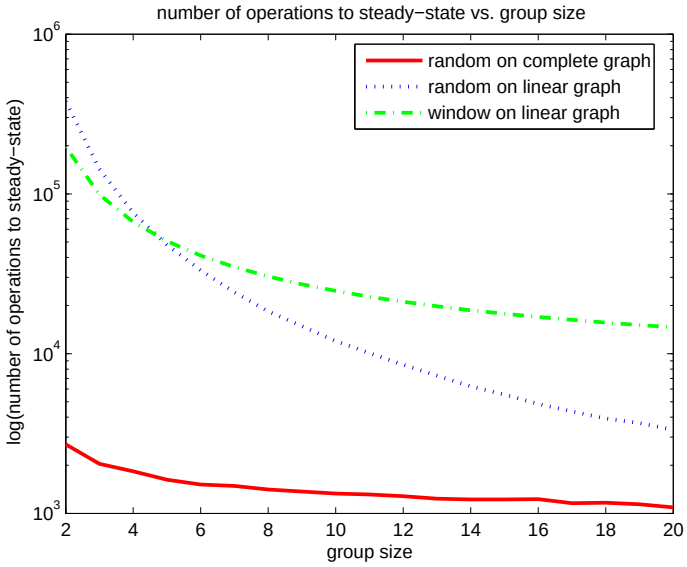


Figure 3.9: Time complexity of the self-similar algorithm for the problem if finding the average of a set of values versus group size assuming the time complexity of a single group operation is linear in group size. We compare the time complexity of the algorithm on a fully connected network with its complexity on a line. When the network is fully-connected, group members are chosen randomly. When agents are on the line we report simulation results for two different group members heuristics. The group size varies between 2 and 20. The y-axis is on a logarithmic scale.

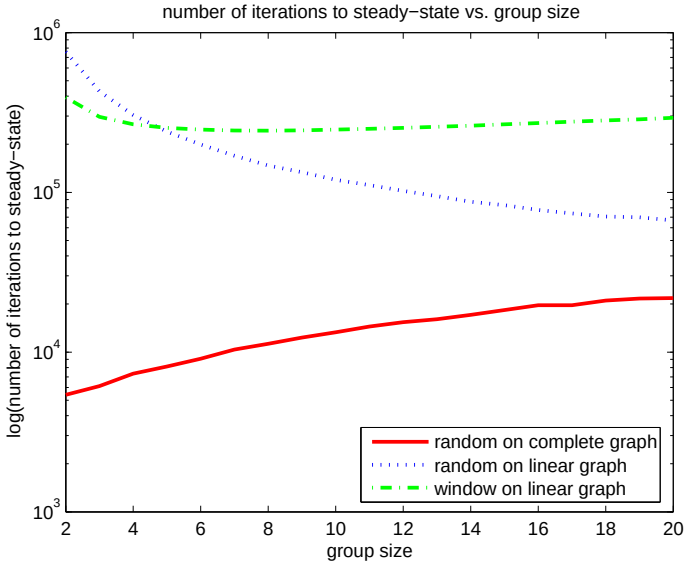


Figure 3.10: Time complexity of the self-similar algorithm for the problem if finding the average of a set of values versus group size assuming the time complexity of a single group operation is proportional to the square of the group size. We compare the time complexity of the algorithm on a fully connected network with the one a line. When the network is fully-connected, group members are chosen randomly. When agents are on the line we report simulation results for two different group members heuristics. The group size varies between 2 and 20. The y-axis is on a logarithmic scale.

3.5.1 A synchronous algorithm for the average problem

Formally, we assume to have a system of N agents where each agent i stores some value V_i. The goal of the system is to compute the average in a distributed fashion. Each agent i updates its state using the following local rule:

$$V_i(t+1) = V_i(t) - \gamma \sum_{j \in N_i} (V_i(t) - V_j(t)) \tag{3.2}$$

where N_i denotes the agents in the neighborhood of agent i, t and $t+1$ are consecutive times. The constant γ describes the rate at which each agent updates its own estimate of the average based on the information from its neighbors. This is a constant which must be less or equals to $\frac{1}{d}$, where d is the degree of the corresponding communication graph. It is easy to show that equation 3.2 is equivalent to the following updating rule:

$$V_i(t+1) = \alpha V_i(t) - (1-\alpha)\frac{1}{|N_i|} \sum_{j \in N_i} (V_j(t)) \tag{3.3}$$

This rule can be expressed as follows: the new value of the agents is a linear combination of the old value of the agent and the average of the values of its adjacents. The value of α is a deterministic function of the value of γ given by $1 - \gamma|N_i|$.

We can combine the local rules to get the global rule expressed in a vectorial notation:

$$V(t+1) = V(t) - \gamma L V(t) \tag{3.4}$$

where L is the Laplacian matrix of the communication graph and it is equal to $D - A$ with D being the degree matrix and A the adjacency matrix of the graph.

The algorithm works as follows. At each iteration, simultaneously, agents collect the values of their agents and update their state. As shown in the paper [24], this algorithm converges for all network configurations. Hence, as the computation precedes, the algorithm converges to the true average.

The algorithm also works when agents and communication links are disabled/enabled, i.e. it is tolerant to environmental attacks.

This approach is different from the idea used by self-similar algorithms. The protocol is completely decentralized but synchronous and each agent makes a local update using values of its adjacent. However, there are some similarities between the two. The local updates of the synchronous algorithm resemble the group idea. The two algorithms work in exactly the same way when the component is fully-connected and α is chosen to be $\frac{1}{k}$, where k is the size of the connected component.

3.5.2 Some comparisons

We briefly discuss and compare the performance of this algorithm with the self-similar one in terms of the number of steps required to converge to the true average. To make the comparison as fair as possible, the simulations are time stepped. At each time step the two algorithms execute a complete step. For the synchronized algorithm, this means that all agents collect the values from their adjacents and update their own values. In the self-similar algorithm, all formed group apply the same optimization step concurrently.

For static connected networks (i.e. networks with a constant communication graph), the self-similar algorithm executes less operations than the synchronous one (except when the network is fully connected when both algorithms converge in just one step). This is because the network consists of only one group; hence, if we execute the self-similar algorithm, all agents store the average in just one step, while the other algorithm needs many more steps to collect the values of distant agents. When the environmental attacks change the structure of the network, the results are different.

As reported in figure 3.11, the number of steps required by the synchronous algorithm to converge (to the true average) decreases as the parameter γ increases. We refer to the paper [24] for a formal proof of this property. This means that we get the best performance of the algorithm when γ is equal to $\frac{1}{d}$ (i.e. the maximum degree of the communication graph). The simulations presented in this figure are based on a fully-connected network of 100 agents. In the figure, we plot the error of the algorithm against the simulation steps for values of γ in $\{0.0001, 0.001, 0.01\}$ where the error is defined as the sum of the squared distance between the current set of values and the true average. When γ is very close to $\frac{1}{d}$ (e.g. $\gamma = 0.01$), the error is very close to 0 as soon as the algorithm takes its first step; hence, the algorithm converges to the true solution in few steps. When γ is very small ($\gamma = 0.0001$), the error starts at 8000 and decreases as the computation proceeds. In this case, the algorithm converges to the true average after 400 steps. When γ is equal to 0.001, the error starts at 6500 and decreases at each iteration converging to 0 after 100 steps of the algorithm.

In figure 3.12, the rate of convergence of the two algorithms in presence of dynamic structures is investigated. On the x-axis there is the size of the group (which varies between 2 and 20) while on the y-axis there is the number of iterations to the steady state for the specific group size in logarithmic scale. In these experiments the network and all subgroups are assumed fully-connected; the size of the network is 100, while at each iterations all groups have the same size k ($k \in [2, 20]$). The synchronous algorithm is executed for several values of γ: $\gamma = \frac{1}{20}$ where 20 is the maximum allowed group size in the simulations, $\gamma = \frac{1}{99}$ where 99 is the maximum degree of the network and $\gamma = \frac{1}{k-1}$ where $k - 1$ is the degree of the current subgroups. From the figure, we have that for dynamics structures the synchronous algorithm outperforms the self-similar algorithm only when sub-group graph are fully connected and at each step the algorithm chooses γ as the degree of the subgraph. If γ is constant (chosen at the beginning of the execution of the algorithm) or the current

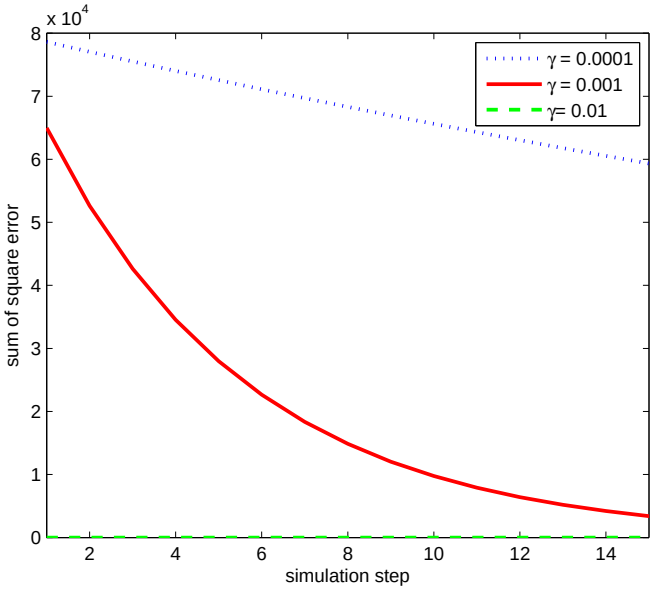


Figure 3.11: Error of the synchronous algorithm as a function of simulation steps for several values of γ. The x-axis represents the number of steps executed by the algorithm while the y-axis represents the distance between the current solution and the true average, which is computed as follows. It is the sum of the square distances between any current value and the actual average. The network is assumed to be fully-connected and of size 100. The parameter γ ranges in the set $\{0.0001, 0.001, 0.01\}$. The maximum degree of the graph is 99. Hence, the maximum value for γ is $\frac{1}{99}$ which is 0.0101.

active subgraph is not fully-connected, the self-similar algorithm works better.

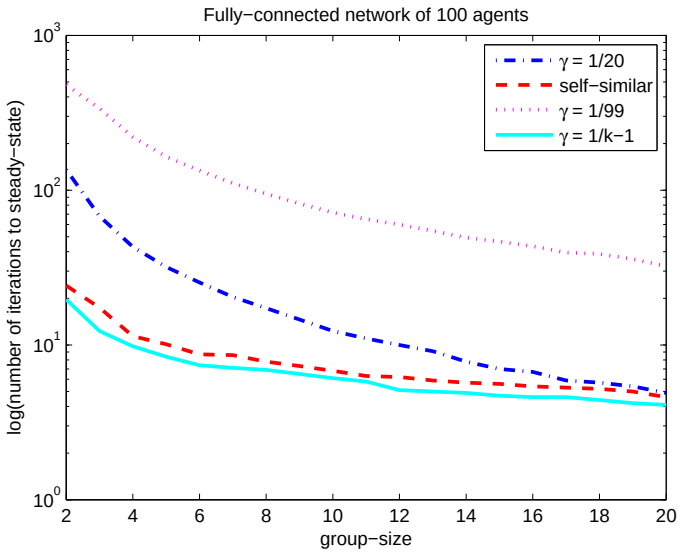


Figure 3.12: Number of iteration of the two algorithm to reach the steady-state as a function of group size. The x-axis represents the group size and the y-axis represents the number of iteration to reach the true average (in a logarithmic scale). The network and all groups are assumed to be fully-connected; the network size is 100. At each iteration the graph is partitioned in groups all of the same size and simultaneously all groups execute the two algorithms. The synchronous algorithm is executed with γ equals to $\frac{1}{20}$, $\frac{1}{99}$ and $\frac{1}{k-1}$ (where k equals to the current size of the group), i.e. $k-1$ is the degree of the subgraphs.

Chapter 4

Self-similar algorithms for non-consensus problems

In this chapter we investigate the performance of self-similar algorithms on a max-min optimization problem, which is not a consensus problem. The task of the agents in the system is to build a "wall" structure. In this chapter, we present the problem in detail, we design and develop a self-similar algorithm to solve the problem and investigate its performance.

4.1 A non-consensus problem: building a wall

Assume to deploy N agents along a line. Each agent i stores a variable V_i containing the amount of resources available at the agent position. Denote by C the amount of total resources available in the system. The task of the agents in the system is to build a wall with the available resources.

When there are no constraints on the height of the wall at some positions, agents exchange resources and eventually all agents have the same amount of resources. Hence, they build a wall with the same height at all locations; throughout this chapter, we denote by C^* this height where C^* is the average amount of resources in the system. A possible instance of the problem with the corresponding final structure is depicted in figure 4.1, where the bars represent the amount of structure available at each agent position. This problem can be reformulated as the problem of finding the average; hence, it belongs to the class of consensus problem and can be solved using the same technique of the previous chapter.

Assume, instead, to constrain the height of the wall at some of the agent locations. Some agents cannot build portions of wall larger than some threshold V, even if the amount of resources available at their position is larger. The remaining amount of resources at these positions should not remain unused, but should be moved and distributed across other agents.

Formally, we assume to deploy two types of agents: the unconstrained ones (denoted by a_U) and constrained ones (denoted by a_C). We use n_U (respectively n_C) to denote the number of

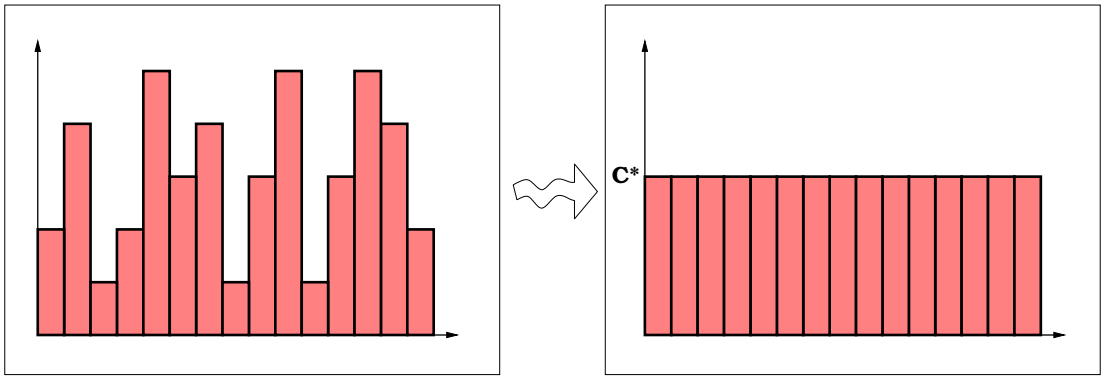


Figure 4.1: An instance of the problem of building a wall when there are no constraints on the amount of resources which can be used by agents. On the left part, the bars represent the initial amount of resources of the agents. By exchanging resources, they eventually reach a state where all agents have the same amount of resources. Hence, they build a wall which have the height C^* at all agent positions.

unconstrained (constrained) agents in the system. The goal of the system is to build a wall as tall as possible wherever possible (for the unconstrained agents) and not taller than V in the constrained positions. This more general formulation of the problem reduces to a consensus-type max-min optimization problem when $C^* \leq V$, where C^* is the height of the wall in the unconstrained corresponding problem. However, the problem is not a consensus problem when $C^* > V$. The goal of the constrained agents is to reach V, while the goal of the unconstrained ones is to move towards the average of the remaining resources $\frac{C-n_C V}{n_U}$, assuming to fully utilize and distribute the latter. An instance of the problem when $V < C^*$ is shown in figure 4.2, where unconstrained agents are shown in red and constrained agents are in blue. In the left part of the figure, the bars represent the initial amount of resources available at each agent while the right part shows the final amounts; constrained agents own V resources while the unconstrained own $\frac{C-n_C V}{n_U}$.

The state of the unconstrained agents consists of the variable V_i which contains the amount of resources available for that agent. Constrained agents have the same variables as the unconstrained ones with the addition of the variable V, which stores the maximum allowed height of the wall at the agent position.

The utility function of the unconstrained agents is the identity function while the utility function for the constrained agents is the identity function for values less than the threshold V and become a straight line for values larger than the threshold. Formally, the utility function of the constrained agents is:

$$f_i(V_i) = \begin{cases} V_i & \text{if } V_i \leq V \\ V & \text{otherwise} \end{cases} \tag{4.1}$$

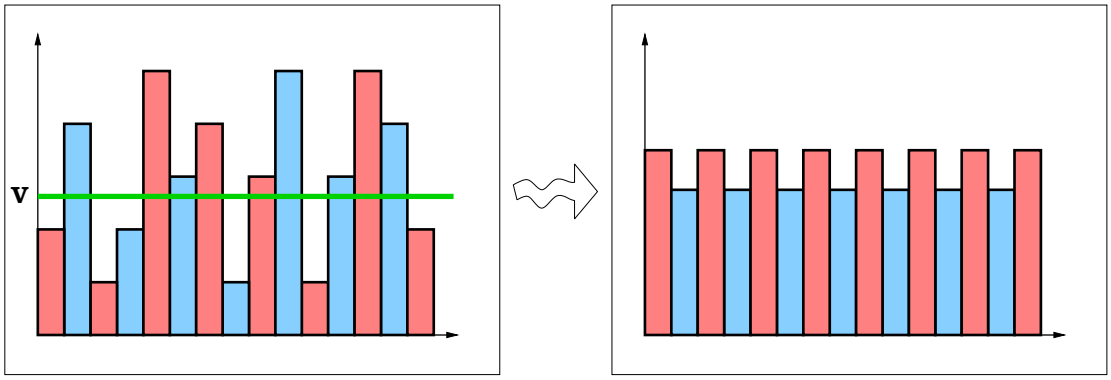


Figure 4.2: An instance of the problem of building a wall when the height of the wall at some agent positions is constrained to be V. Unconstrained agents are shown in red while constrained are shown in blue. The initial distribution of resources is shown in the left part of the figure while the final configuration is shown on the right part. At the steady-state the wall in correspondence of blue agents has height V.

4.2 A self similar algorithm for wall problem

In the self-similar technique agents try to solve the global problem by improving the group local solutions, in such a way that local improvements are compatible with global improvements while maintaining group constraints.

For this problem, the action taken by the groups depends on the structure of the group:

- If all agents of the group have the same type, they compute the average of their values and update their state with this new value:

$$V_i(t) = \frac{1}{|G|} \sum_{j \in G} V_j(t-1) \tag{4.2}$$

- When the group is mixed, i.e. it consists of constrained and unconstrained agents, and the subproblem can be reduced to a consensus problem ($C^*_G \leq V$ where C^*_G is the average amount of resources of the group), they update their state with the value of their average:

$$V_i(t) = \frac{1}{|G|} \sum_{j \in G} V_j(t-1) = C^*_G(t-1) \tag{4.3}$$

- When the group is mixed and the subproblem is not a consensus problem, different actions are taken by different agents. The subgroup consisting of constrained agents modifies its state by setting its state to V. Hence, the value V_i of agent $i \in G_C$ at the current time t is

$$V_i(t) = V \tag{4.4}$$

Denote by

$$Res_C(t) = \sum_{j \in G_C} (V_j(t-1) - V) \tag{4.5}$$

the residue of the group at time t. If this value is positive, constrained agents have some unused resources which should be redistributed among the unconstrained agents. If it is negative, then constrained agents need more resources and need to steal some of them from the unconstrained ones to reach the amount V. Hence, the unconstrained agents should set their values to the average between the sum of their current values and the residue of the constrained ones in order to satisfy the conservation low:

$$V_i(t) = \frac{1}{|G_U|}\Big(\sum_{j \in G_U} (V_j(t-1)) + Res_C(t) \Big) \tag{4.6}$$

where G_U is the subgroup of unconstrained agents.

The self-similar algorithm converges to a steady-state which is a straight wall (when the average amount of resources is less than V) and the asymmetric wall structure explained in the previous section (when more resources are available). It is easy to generalize this self-similar algorithm to deal with any possible wall structure. This can be done by further categorizing constrained agents to get different values.

4.3 Investigating the performance of the algorithm

In this section, we investigate the performance of the self-similar algorithm. We try to answer the following questions:

- Is the total amount of available resources related to the rate of convergence? We want to understand whether there is a linear relationship between the time needed to converge and the amount of resources.
- Is the number of steps required to converge to the steady-state related to the threshold value V of the constrained agents?
- How do the network agent composition and constrained agent deployment affect the rate of convergence? Does the algorithm converge faster when the majority of agents is constrained, or when is it unconstrained? Is it better to deploy the constrained agents close to each other than just randomly?

All experiments are based on 100 MonteCarlo simulation on a fully-connected network of 50 agents. Agent initial values are random in the interval $[0, 100]$.

4.4 Impact of amount of total resources

In figure 4.3, we investigate the relation between the number of group operation (needed to converge to the steady-state) and the amount of available resources in the system. In this experiment, we assume the network to be fully-connected of size 50 with 25 unconstrained agents and 25 constrained (at value $V = 40$). In the figure, on the x-axis there is the amount of resource varying between 500 and 4000, while on the y-axis is reported the number of group operations required to reach the steady-state in logarithmic scale. The results of the experiments are shown for several group size (group of size 2, 5, 10, 15 and 20). When $C \leq 2000$, the problem reduces to a consensus problem, while when it is larger than 2000 it is not a consensus problem anymore. From the figure we can see that the consensus formulation has different time complexity from the non-consensus one. When $C < 2000$, the convergence time of the algorithm is slightly increasing, but smaller than 10^3. This implies that if we add more resources to the problem, the time complexity increases. We have the same behavior for all group sizes. Instead, when C is larger than 2250, it is slightly decreasing, but larger than 10^3. Finally, the time complexity of the case when C is 2000 is different from the two subcases and fills the gap between the two. It's value is around 10^3.

Our intuition can suggest that the time complexity should not depend on specific initial sets of values if their assignment is completely random. This anomaly can be explained as follows. In our case initial values of the agents are not really completely random. This is because we are not allowing negative values and initial values are chosen in the interval $[0, 100]$. When the overall sum is either very small or very large, the set of feasible values for each agent is very small, and the resulting values are very close to each other. Instead, when the sum is in the middle ($C = 2000$), agent initial values can vary in large set and hence be far away from each other. Hence, for C small, as the number of resources increases, agents values get far away and thus more time is needed to reach consensus. When $C = 2000$, agents values can be considered to be chosen according to a uniform distribution. When C keeps increasing the interval of feasible values starts to become smaller again. Hence, the time complexity decreases even if the non-consensus formulation takes more time than the consensus one.

4.5 Impact of maximum wall height

Using the same settings as in the previous experiment, we investigate how the maximum allowed height of the constrained agents affects the rate of convergence of the algorithm. The network consists of 25 constrained and 25 unconstrained agents. The value of V varies between 10 and 60 and the total amount of resources is constant and its value is 2500. Under these parameter settings, we have the problem reduces to a consensus problem if $V \geq 50$; instead, when $V < 50$, the problem

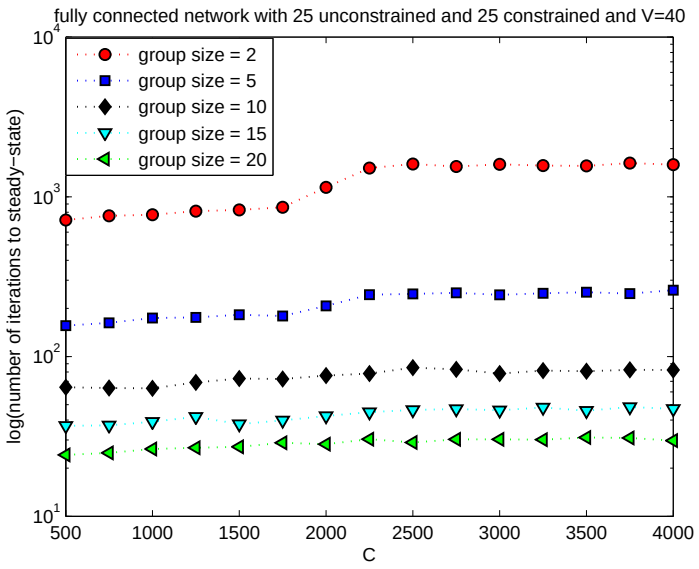


Figure 4.3: Time complexity of the self-similar algorithm for the wall problem versus the amount of resources available. The network is fully connected consisting of 25 unconstrained agents and 25 constrained. The simulation results are shown for several group sizes. The value of V is 40.

becomes a non-consensus problem.

The results are shown in figure 4.4. The x-axis represents the size of the group (varying between 2 and 20) and the y-axis represents the number of group meetings needed to reach consensus in logarithmic scale. The several curves in the plot refer to the time complexity of the algorithm assuming V in the range between 10 and 60. As our intuition suggests, the algorithm based on a consensus formulation outperforms the same algorithm based on a non consensus formulation. As the value V increases, the time complexity of the algorithm strictly decreases. This is because increasing V implies a smaller amount of resources to distribute across unconstrained agents. We have the same behavior for all group sizes.

4.6 Impact of constrained agents

In this section we investigate two questions about constrained agents: (1) does the number of constrained agents affect the rate of convergence? (2) do the positions of constrained agents affect the rate of convergence?

For the first set of experiments, we consider a fully-connected network of 50 agents and study the frequency of constrained agents which varies between 0 and 49. The value of V is set to 40. We run two different experiments: the first one depicted in figure 4.5 refers to the case when $C = 1500$and the second to the case when $c = 2500$ (see 4.6). On the x-axis of these figures there is the group size which varies between 2 and 20, while on the y-axis there is the number of steps needed to the

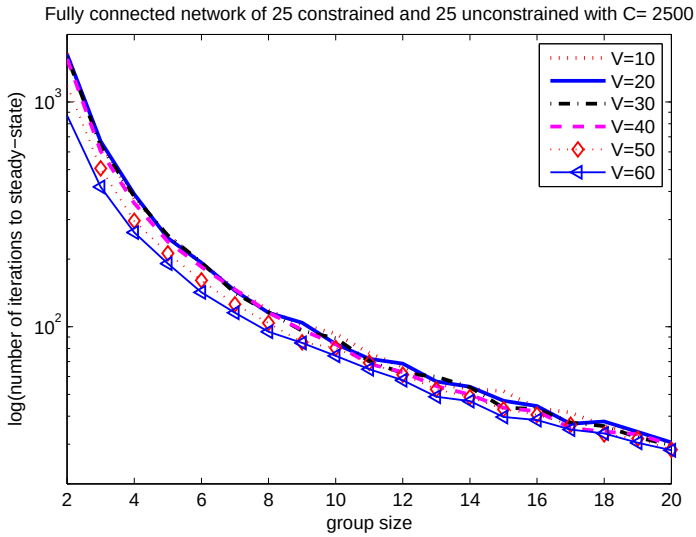


Figure 4.4: Time complexity of the self-similar algorithm for the wall problem versus group size varying the maximum allowed height for constrained agents. The network is fully connected consisting of 25 unconstrained agents and 25 constrained and the total amount of resources is set to 2500. The maximum height varies between 10 and 60.

algorithm to converge to the steady-state in logarithmic scale.

In the first experiment ($C = 1500$), the problem reduces to a consensus problem. In this case, we can see from the figure that regardless of the composition of the network, the time complexity is always the same, i.e. it does not depend on the number of constrained agents. This is because we are solving the same optimization problem regardless the specific number of constrained agents. Instead, in the second experiment, the amount of resources is 2500 and the results are completely different. In this case, as intuition suggests, the time complexity of the algorithm increases as more constrained agents are added to the network, because we are adding more constraints to the optimization problem.

In the last experiment we investigate the second question, i.e. we study whether positions of constrained agents affects the time complexity of the algorithm. For this experiment we cannot assume the underlined network to be a fully connected graph. This is because for this structure any chosen position is equivalent to any other. In this experiment, the communication graph is assumed to be a line, i.e. each agent can communicate directly only with its left and right adjacents and the formed groups can only be only segments consisting of consecutive agents. The number of agents is 50 with 25 constrained and 25 unconstrained. We compare three possible cases: (1) constrained agents are randomly distributed around unconstrained, (2) constrained agents are consecutive, (3) constrained agents are interleaved between unconstrained (i.e. $a_C a_U a_C a_U \ldots$). The total amount of resources is 2500 and the value of V is 40, i.e. we are solving a non-consensus instance of the problem. The results are depicted in figure 4.7. As intuition suggests, the random and interleaved

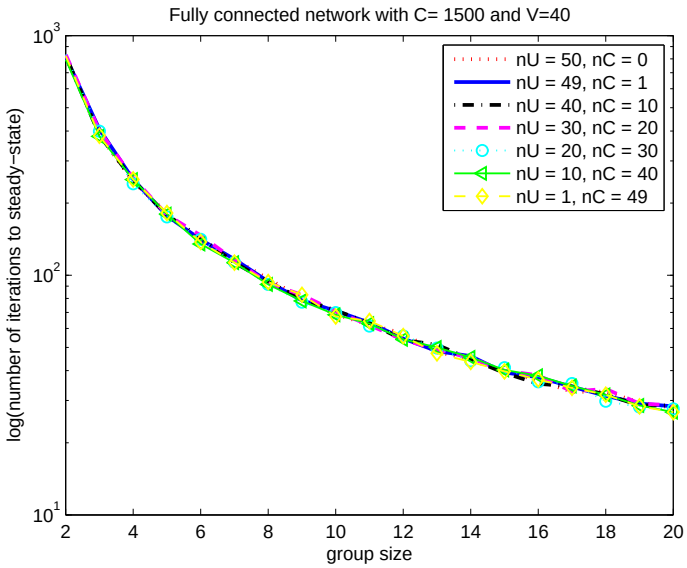


Figure 4.5: Time complexity of the self-similar algorithm for the wall problem versus group size varying the number of constrained agents in the network. The network is fully connected with total amount of resources equals to 1500 and maximum allowed height is 40.

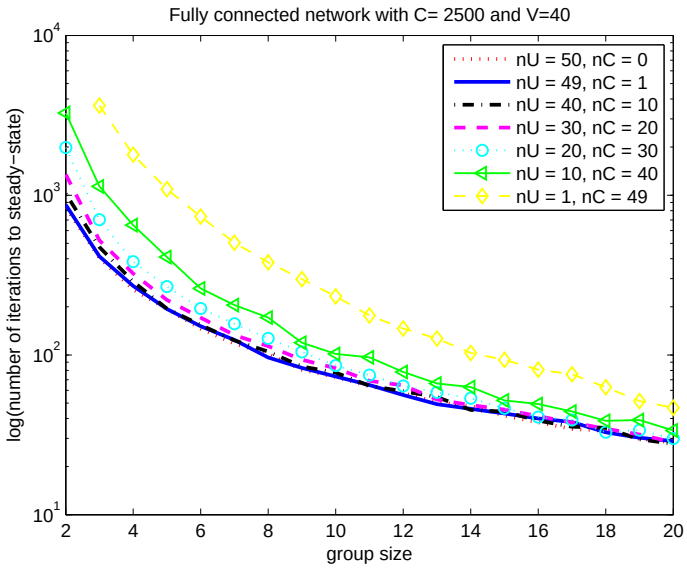


Figure 4.6: Time complexity of the self-similar algorithm for the wall problem versus group size varying the number of constrained agents in the network. The network is fully connected with total amount of resources equals to 2500 and maximum allowed height is 40.

deployments outperform the other consecutive deployment. When agents are consecutive we get the worst performance: this is because mixed groups are formed very rarely and, hence, the communication between different types of agents is very slow. When, constrained agents interleaves between unconstrained agents, we have that in all formed group there is the number number of constrained and unconstrained agents. On one hand, this speeds up the algorithm, because constrained agents are more likely to set their values to V; on the other hand, it slows down the algorithm, because unconstrained agents need more time to distribute their values, making the algorithm slower than the algorithm on a random deployment.

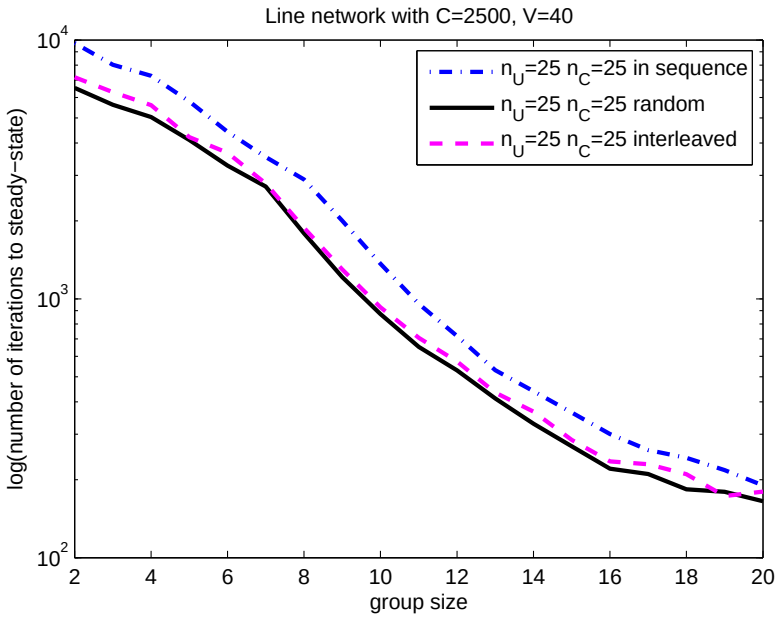


Figure 4.7: Time complexity of the self-similar algorithm for the wall problem versus group size varying the deployment of constrained agents. The network is a line of size 50 with 25 constrained and 25 unconstrained agents; the total amount of resources equals to 2500 and maximum allowed height is 40.

4.7 Variation of the problem

In the problem of building a wall, the system has three goals: (1) all constrained agents should build a portion of the wall with height at most V, (2) resources should be fully utilized and (3) all unconstrained agents should use the same amount of resources.

In this section, we relax some of these constraints. In particular, assume that we are not interested in a the specific final structure where unconstrained agents build structures at the same level. In the variation we want to discuss, unconstrained agents do not have to agree on some specific quantity. The task of the agents in the system is to build a wall with height at least V and to fully utilize the

resources (assuming enough resources, i.e. $C^* > V$). This means that while constrained agents will create portion of the wall of height V, unconstrained ones will build portion of the wall of arbitrarily height, but larger than V.

This problem does not have a unique solution: any wall of height at least V which satisfies the conservation law and the restrictions on constrained agents is a feasible solution. Applying a self-similar technique, the solution depends on the sequence of group meetings.

When the average amount of resource $C^* \leq V$, the problem can be formulated as a consensus problem, where all agents agree on the quantity C^*.

Each agent i stores the variable V_i which indicates the current amount of resources available at its position and the value V. The utility functions of the unconstrained and constrained agents are respectively the identity function and the function in equation 4.1.

4.8 A self-similar algorithm for the problem variation

When a group meets, agents perform different tasks:

- if all agents are of the same type, then they compute the average of their values.
- if the group is mixed and the subgroup problem is a consensus problem, then agents take the average of the values.
- if the group is mixed, the problem is not consensus and the exit conditions are not satisfied. All constrained agents are set to V and the unconstrained one are moved to the value given by equation 4.6.
- in the remaining case, the group does not execute any operation.

This algorithm converges to some steady state where the constraints are satisfied. The solution to this problem can be far away from a smooth and nice wall. However, as it will be shown in the next section, it converges much faster than the previous algorithm.

4.9 Performance of the algorithm

We discuss some simulation results for this algorithm and investigate the same questions presented in section 4.3.

We investigate how the amount of resources is related to the rate of convergence of the algorithm. The network is assumed to be fully-connected of size 50 with only one constrained agent ($V = 40$). The results are shown in figure 4.8 for several group sizes. On the x-axis there is the amount of available resources which varies between 500 and 3000, while on the y-axis there is the number of

steps to the steady-state. The curves in the figure refer to group size equal to 2, 5, 10 and 20. When $C \leq 2000$, the problem reduces to a consensus problem, while when it is larger than 2000 it is not a consensus problem anymore. From the figure we can see that the consensus formulation has different time complexity from the non-consensus one. When $C < 2000$, the convergence time of the algorithm is increasing, starting from 700 and reaching 900. This implies that if we add more resources to the problem, the time complexity increases. We have the same behavior for all group sizes. Instead, when C is larger than 2000, it decreases very fast, assuming values between 10 and 300. This anomaly depends on the initial values given to the agents which are not (and cannot be) really random if we fix C. When C is very small or very large, agents can assume values in very small intervals; hence as we increase C, the set of feasible values increases until $C = 2000$. When $C = 2000$ initial values are really random and as C continues increasing the set of admissible values decreases; hence, the algorithm converges faster. However, the algorithm on a non consensus instance outperforms the algorithm on a consensus instance. This is because when C becomes large, the initial values assigned to the agents get larger, and they may become larger than V in a few steps, thereby satisfying the requirements.

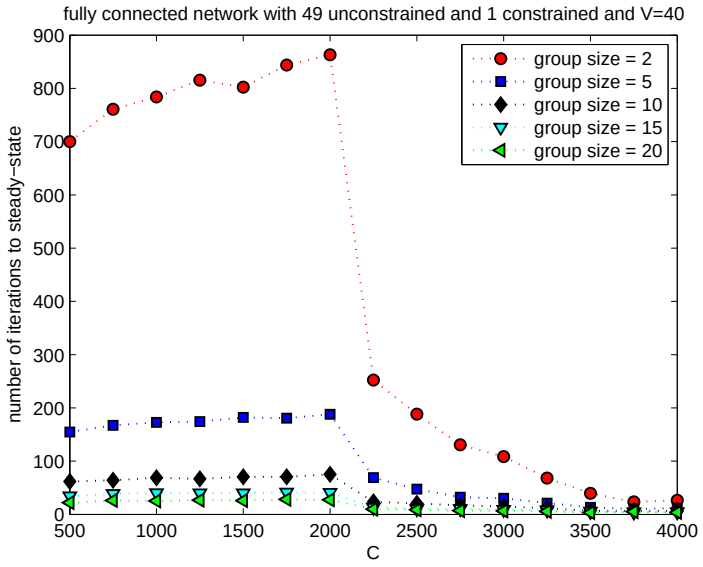


Figure 4.8: Time complexity of the self-similar algorithm for the variation of the wall problem versus the amount of resources available. The network is fully connected consisting of 49 unconstrained agents and 1 constrained one. The simulation results are shown for several group sizes and the value $V = 40$.

In figure 4.9, we report experimental results to investigate whether the value of V affects the convergence rate of the algorithm. We simulate a fully-connected network of 50 agents with only 1 constrained agent and the total amount of resources is 2500. On the x-axis there is the size of the group (varying between 2 and 20), while on the y-axis there is the time complexity of the algorithm.

The curves in the figure refer to different values of V which varies between 10 and 60. When $V < 50$, the instance gives a non consensus solution and the time complexity of the algorithm increases with V. This is because assuming V small and C large, we have that many agents store values above the threshold V. As V increases we need more steps to satisfy this constraint. Instead, when $V \geq 50$, the instance reduces to a consensus problem and hence does not depend on the value of V. The time complexity of the algorithm for a consensus instance increases moving from 10^2 to 10^3 for a group of size 2.

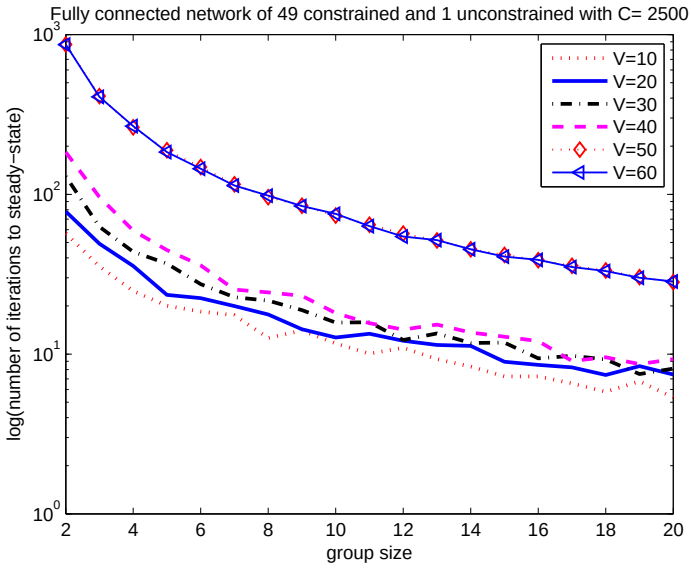


Figure 4.9: Time complexity of the self-similar algorithm for the variation of the wall problem versus group size varying the maximum allowed height for constrained agents. The network is fully connected consisting of 49 unconstrained agents and 1 constrained and the total amount of resources is set to 2500. The maximum height varies between 10 and 60.

In figure 4.10 and 4.11, we investigate whether the rate of convergence of the algorithm is related to the number of constrained agents. In the experiment, the network is assumed to have size 50 and the threshold value V is set to 40. The x-axis represents the group size while the y-axis represents the number of steps that the algorithm needs to converge to a steady-state solution in logarithmic scale. When the total amount of resources is 1500, the instance of the problem reduces to a consensus problem; in this case, the time complexity does not depend on the fraction of constrained agents. When the problem is a non consensus ($C = 2500$), as intuition suggests, the time complexity of the algorithm increases if we add more constrained agents. This is because the number of constraints to be satisfied increases.

Finally, we consider how the position of the constrained agents affects the time complexity of the algorithm. As in section 4.6, we assume the underlying communication graph to be a line and investigate three possible deployments: constrained agents are consecutive, constrained agents are

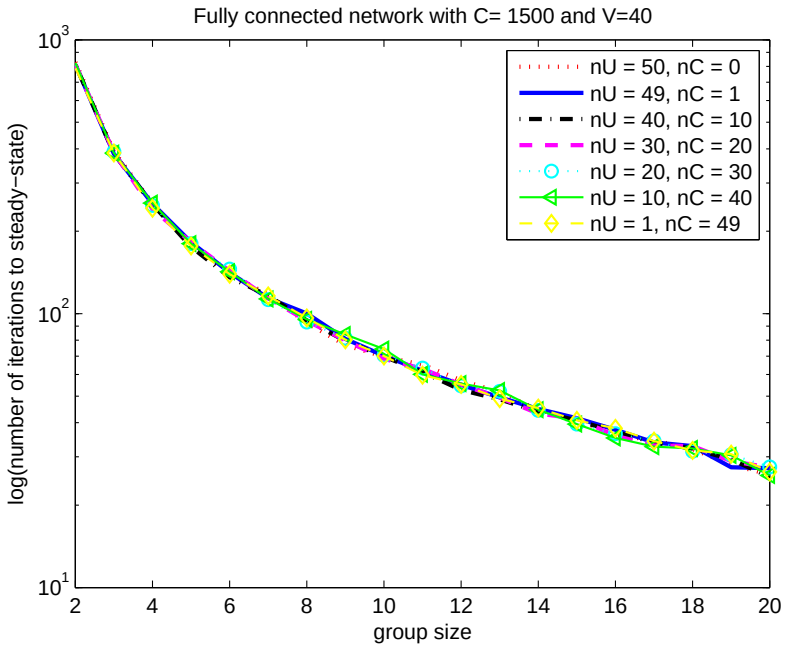


Figure 4.10: Time complexity of the self-similar algorithm for the wall problem versus group size varying the number of constrained agents in the network. The network is fully connected with total amount of resources equals to 1500 and minimum allowed height sets to 40.

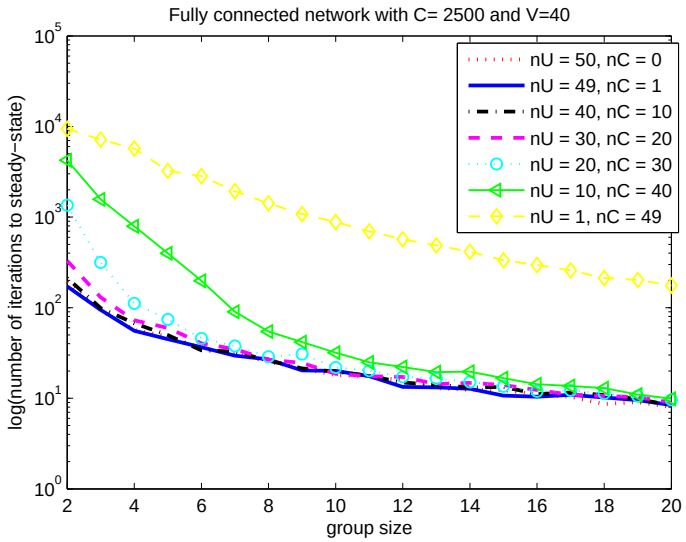


Figure 4.11: Time complexity of the self-similar algorithm for the variation of the wall problem versus group size varying the number of constrained agents in the network. The network is fully connected with total amount of resources equals to 2500 and minimum allowed height sets to 40.

randomly deployed and constrained agents are interleaved with unconstrained ones. In this case, the interleaved and random deployments outperform the consecutive deployment. This is because in consecutive deployments the communication between constrained and unconstrained agents is very rare. The results of this experiment are displayed in figure 4.12.

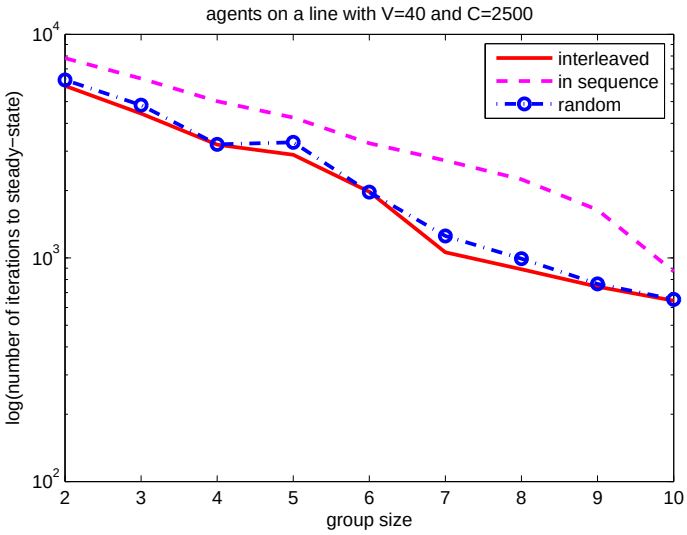


Figure 4.12: Time complexity of the self-similar algorithm for the variation of the wall problem versus group size varying the deployment of the constrained agents in the network. The network is a line of size 50 with 25 constrained and 25 unconstrained agents; the total amount of resources is 2500 and $V = 40$.

Chapter 5

A synchronous algorithm for mobile agent formations

This chapter investigates coordination problems where the goal of agents in the system is to move and reach some specific configuration. In particular, we focus on a generalization of the equidistance problem. We design and implement a synchronous algorithm for this problem and prove its correctness using techniques from distributed systems and control theory.

5.1 Generalized equidistance problem

We have discussed the equidistance problem in section 1.7. We have shown that when agents are on a line and agents cannot pass each other, this problem can be reduced to the problem of finding the average. Hence, for the equidistance problem we can apply the self-similar algorithm discussed in the previous chapter and obtain the solution.

In this chapter, we consider a generalization of the equidistance problem where agents deployed along a line are allowed to pass each other. This changes the specification of the problem; the problem cannot be expressed as a max-min optimization problem any longer. This is because we cannot find a formal definition for the problem which satisfies both constraints simultaneously.

In this chapter we design a synchronous algorithm which is similar to the algorithm presented in the paper [24] for solving the problem of the average.

The problem can be formally stated as follows. A set of $N + 1$ agents is randomly distributed over a line, each associated with a unique identifier. Agents 0 and N remain fixed at position 0 and C; while all other agents can move. The goal of the mobile agents is to be placed equidistantly on the line.

The agent communication protocol is based on agent ids, rather than agent positions and distances. This can be done by assuming very large transmission ranges. Each agent stores a set of ids; when it receives a message from one of these agents, it processes the message and discards it

otherwise. Differently from a distance based protocol, the communication between agents is not necessarily bi-directional. For this specific problem we assume that each mobile agent i (for $0 < i < N$) can receive messages only from agent $i-1$ and $i+1$.

The state of an agent consists of its id, its current position stored in the variable x_i and the list of agents which are allowed to send it messages.

5.2 Modeling the problem using directed graphs

The communication graph G of the system is a directed graph (i.e. a graph with oriented edges) where there is an edge orientated from vertex v to w if and only if agent v can receive messages from agent w. The graph remains unchanged during the computation; this is because it depends only on the agent communication lists which cannot be modified.

The adjacency matrix $A(G)$ has rows consisting of zero entries in correspondence of stationary agents. The nonzero entries of mobile agents are instead assigned according to their lists: for each agent i (with $0 < i < N$) only the two entries $(i, i+1)$ and $(i, i-1)$ are set to 1 in the row i, while all other entries are 0.

We denote by D the out-degree matrix of G, and by L the Laplacian matrix of G which is defined as follows:

$$L = D - A \tag{5.1}$$

5.3 Synchronous time-stepped algorithm

We design a synchronous time-stepped algorithm where each agent computes its own estimate of its true position based on the estimates of its adjacents. Similarly to the synchronous time-stepped algorithm in [24], each agent i updates its state using the following local rule. The new position of agent i is given by:

$$x_i(t+1) = x_i(t) - \gamma \sum_{j \in N_i} (x_i(t) - x_j(t)) \tag{5.2}$$

where N_i denotes the agents in the neighborhood of agent i, t and $t+1$ are consecutive times and $0 < \gamma < \frac{1}{d}$, where d is the degree of the corresponding communication graph. By setting $\alpha = 1 - \gamma |N_i|$ for $0 < \alpha < 1$, we find an equivalent updating rule:

$$x_i(t+1) = \alpha x_i(t) + (1-\alpha)(\sum j \in N_i x_j)/|N_i| \tag{5.3}$$

which informally states that the new position of each agent can be computed as the linear combina-

tion of its current position and the average of the current positions of its adjacents. Instead, when the agent is isolated, it remains in the same position.

We can combine the local rules to get the global rule expressed in a vectorial notation:

$$x(t+1) = x(t) - \gamma L x(t) \tag{5.4}$$

where L is the Laplacian matrix of the communication graph of the system.

The algorithm works as follows. At each iteration, simultaneously, agents collect the values of their adjacents and update their state. Differently from the algorithm in [24] (implemented for computing the average), we allow directional communication. The algorithm works also when agents and communication links are disabled/enabled, i.e. it is tolerant to environmental attacks.

5.4 Proof of convergence using distributed system techniques

As the computation proceeds, applying the rule in equation 5.2 (or equation 5.3), agents eventually reach some steady configuration. We want to prove that this final configuration is indeed the equidistance configuration.

The proof of correctness and termination of the algorithm proceeds as follows:

- we define a function h;
- we prove that it is bounded below;
- we show that at each iteration of the algorithm which changes the state of the system, the function h decreases.

This is a common technique in distributed system for proving correctness and termination; the function h is usually called a *variant function*.

The function h is defined as follows:

$$h(t) = \sum_{j=1}^{N} (x_j(t) - x_{j-1}(t))^2 \tag{5.5}$$

This is the sum of the squared distances between two consecutive agents. Throughout the section, we denote by $\delta_j(t) = x_j(t) - x_{j-1}(t)$ to simplify the notation. Hence, we can rewrite the function as follows:

$$h(t) = \sum_{j=1}^{N} \delta_j^2(t) \tag{5.6}$$

Without loss of generality, we assume C (i.e. the position of agent with id N) to be 1. Due to the constraints on the stationary agent positions, which are at positions 0 and 1, we can show that

Proposition 1 *If the positions of agents are updated according to equation 5.2, we get that* $h(t) \geq \frac{1}{N}$ *with equality if and only if for all* i *the position of* $x_i = \frac{i}{N}$.

It remains to prove that the variant function decreases whenever the state of the system changes:

Theorem 2 *If* $x(t+1) \neq x(t)$ *then* $h(t+1) < h(t)$

The values of the x_i are modified according to the following rules which can be derived from equation 5.3:

- $x_0(t+1) = x_0(t)$
- $x_N(t+1) = x_N(t)$
- $x_i(t+1) = \alpha x_i(t) + (1-\alpha)\frac{x_{i-1}(t)+x_{i+1}(t)}{2}$

The proof of the theorem 2 proceeds as follows. We compute an expression for $h(t+1)$ in terms of $x_i(t)$ and we solve for the difference between $h(t)$ and $h(t-1)$ by showing that it is strictly positive. We can rewrite the expression of $h(t+1)$ as the sum of the three following terms:

$$\delta_1^2(t+1) + \sum_{j=2}^{N-1} \delta_j^2(t+1) + \delta_N^2(t+1) \tag{5.7}$$

We evaluate each term and for each of them we compute an equivalent expression in terms of the $x_i(t)$.

Proposition 2 $\delta_1(t+1) = \alpha\delta_1(t) + (1-\alpha)\frac{\delta_1(t)+\delta_2(t)}{2}$

This equivalence can be proven as follows. By substituting the definition of $x_1(t+1)$ and $x_0(t+1)$ in terms of $x_0(t)$ and $x_1(t)$ we obtain that

$$\delta_1(t+1) = \alpha x_1(t) + (1-\alpha)\frac{x_0(t)+x_2(t)}{2} - x_0(t) \tag{5.8}$$

Multiplying $x_0(t)$ by $(\alpha + (1-\alpha))$ we get that

$$\delta_1(t+1) = \alpha x_1(t) + (1-\alpha)\frac{x_0(t)+x_2(t)}{2} - (\alpha + 1 - \alpha)x_0(t) \tag{5.9}$$

and rearranging terms we get

$$\delta_1(t+1) = \alpha(x_1(t) - x_0(t)) + (1-\alpha)\frac{(x_2(t)-x_1(t)) + (x_1(t)-x_0(t))}{2} \tag{5.10}$$

which can be rewritten as

$$\delta_1(t+1) = \alpha\delta_1(t) + (1-\alpha)\frac{\delta_2(t)+\delta_1(t)}{2} \tag{5.11}$$

We can apply the same steps for $\delta_N(t+1)$ to obtain an equivalent expression in terms of the $x_i(t)$.

Proposition 3 $\delta_N(t+1) = \alpha\delta_N(t) + (1-\alpha)\dfrac{\delta_N(t) + \delta_{N-1}(t)}{2}$

We finally get an equivalent expression for $\delta_j(t+1)$ when $1 < j < N$ as follows:

Proposition 4 $\delta_j(t+1) = \alpha\delta_j(t) + (1-\alpha)\dfrac{\delta_{j+1} + \delta_{j-1}}{2}$

This can be shown as follows. Using the expressions for $x_j(t+1)$ and $x_{j-1}(t+1)$ we get:

$$\delta_j(t+1) = \alpha x_j(t) + (1-\alpha)\frac{x_{j-1}(t) + x_{j+1}(t)}{2} - \alpha x_{j-1}(t) - (1-\alpha)\frac{x_{j-2}(t) + x_j(t)}{2} \tag{5.12}$$

Rearranging the terms, we obtain

$$\delta_j(t+1) = \alpha(x_j(t) - xj - 1(t)) + \frac{(1-\alpha)}{2}(x_{j+1}(t) - x_j(t) + x_{j-1}(t) - x_{j-2}(t)) \tag{5.13}$$

which becomes

$$\delta_j(t+1) = \alpha\delta_j(t) + (1-\alpha)\frac{\delta_{j+1}(t) + \delta_{j-1}(t)}{2} \tag{5.14}$$

Putting together propositions 2, 3 and 4 and computing the squares, we can rewrite the function $h(t+1)$ in terms of the x_i as follows:

$$\begin{aligned} h(t+1) \quad = \quad & \textstyle\sum_{j=1}^{N}\left(\alpha^2 + \frac{(1-\alpha)^2}{4} + \frac{(1-\alpha)^2}{4}\right)\delta_j^2(t) + \\ & +\alpha(1-\alpha)\left(2\textstyle\sum_{j=1}^{N-1}\delta_j(t)\delta_{j+1}(t) + (\delta_1^2(t) + \delta_N^2(t))\right) + \\ & +\frac{(1-\alpha)^2}{2}\left(\delta_2(t)\delta_1(t) + \delta_N(t)\delta_{N-1}(t) + \textstyle\sum_{j=2}^{N-1}\delta_{j+1}(t)\delta_{j-1}(t)\right) \end{aligned} \tag{5.15}$$

We can now compute the difference between $h(t)$ and $h(t+1)$ which is equal to

$$\begin{aligned} h(t) - h(t+1) \quad = \quad & \textstyle\sum_{j=1}^{N}\left(\frac{1 + 2\alpha - 3\alpha^2}{2}\right)\delta_j^2(t) + \\ & -\alpha(1-\alpha)\left(2\textstyle\sum_{j=1}^{N-1}\delta_j(t)\delta_{j+1}(t) + (\delta_1^2(t) + \delta_N^2(t))\right) + \\ & -\frac{(1-\alpha)^2}{2}\left(\delta_2(t)\delta_1(t) + \delta_N(t)\delta_{N-1}(t) + \textstyle\sum_{j=2}^{N-1}\delta_{j+1}(t)\delta_{j-1}(t)\right) \end{aligned} \tag{5.16}$$

We notice that

$$(\delta_j(t) - \delta_{j+1}(t))^2 = \delta_j^2(t) - 2\delta_j(t)\delta_{j+1}(t) + \delta_{j+1}^2(t) \tag{5.17}$$

We can rearrange the terms by putting together the cross terms with the square values. We need a number of terms equals to $\alpha(1-\alpha)$ for $\delta_1^2(t), \delta_N^2(t)$ and $2\alpha(1-\alpha)$ for $\delta_j^2(t)$

The equation becomes:

$$\begin{aligned} h(t) - h(t+1) &= \sum_{j=1}^{N} \left(\tfrac{1+2\alpha-3\alpha^2}{2} - 2\alpha(1-\alpha) \right) \delta_j^2(t) + \\ &+ \alpha(1-\alpha) \left(\sum_{j=1}^{N-1} (\delta_j(t) - \delta_{j+1}(t))^2 \right) + \\ &- \frac{(1-\alpha)^2}{2} \left(\delta_2(t)\delta_1(t) + \delta_N(t)\delta_{N-1}(t) + \sum_{j=2}^{N-1} \delta_{j+1}(t)\delta_{j-1}(t) \right) \end{aligned} \tag{5.18}$$

But

$$\frac{1+2\alpha-3\alpha^2}{2} - 2\alpha(1-\alpha) = \frac{(1-\alpha)^2}{2} \tag{5.19}$$

We apply the same trick for $\delta_1(t)\delta_2(t)$ and we need $\frac{(1-\alpha)^2}{4}$ terms with $\delta_1^2(t)$ and $\delta_2^2(t)$. Hence, the difference becomes:

$$\begin{aligned} h(t) - h(t+1) &= \frac{(1-\alpha)^2}{4} \sum_{j=1,2} \delta_j^2(t) + \frac{(1-\alpha)^2}{2} \sum_{j=3}^{N} \delta_j^2(t) + \\ &+ \alpha(1-\alpha) \left(\sum_{j=1}^{N-1} (\delta_j(t) - \delta_{j+1}(t))^2 \right) + \frac{(1-\alpha)^2}{4} (\delta_1(t) - \delta_2(t))^2 + \\ &- \frac{(1-\alpha)^2}{2} \left(\delta_N(t)\delta_{N-1}(t) + \sum_{j=2}^{N-1} \delta_{j+1}(t)\delta_{j-1}(t) \right) \end{aligned} \tag{5.20}$$

We apply again the same trick for $\delta_N(t)\delta_{N-1}(t)$. We need $\frac{(1-\alpha)^2}{4}$ terms with $\delta_N^2(t)$ and $\delta_{N-1}^2(t)$, thus the difference becomes

$$\begin{aligned} h(t) - h(t+1) &= \frac{(1-\alpha)^2}{4} \sum_{j=1,2,N-1,N} \delta_j^2(t) + \frac{(1-\alpha)^2}{2} \sum_{j=3}^{N-2} \delta_j^2(t) + \\ &+ \alpha(1-\alpha) \left(\sum_{j=1}^{N-1} (\delta_j(t) - \delta_{j+1}(t))^2 \right) + \\ &+ \frac{(1-\alpha)^2}{4} \sum_{j=2,N} (\delta_j(t) - \delta_{j-1}(t))^2 + \\ &- \frac{(1-\alpha)^2}{2} \left(\sum_{j=2}^{N-2} \delta_{j+1}(t)\delta_{j-1}(t) \right) \end{aligned} \tag{5.21}$$

We complete the remaining cross product and we need $\frac{(1-\alpha)^2}{4}$ terms with $\delta_j^2(t)$ for $j \in \{1, 2, N-1, N\}$ and $\frac{(1-\alpha)^2}{2}$ for the others. This step balances out the sum of the $\delta_j^2(t)$ and we get

$$\begin{aligned} h(t) - h(t+1) &= \alpha(1-\alpha)\Big(\textstyle\sum_{j=1}^{N-1}(\delta_j(t) - \delta_{j+1}(t))^2\Big) + \\ &+ \frac{(1-\alpha)^2}{4}\textstyle\sum_{j=2,N}(\delta_j(t) - \delta_{j-1}(t))^2 + \\ &+ \frac{(1-\alpha)^2}{4}\Big(\textstyle\sum_{j=2}^{N-2}(\delta_{j+1}(t) - \delta_{j-1}(t))^2\Big) \end{aligned} \tag{5.22}$$

Assuming that $0 \leq \alpha \leq 1$, we get that both $\alpha(1-\alpha)$ and $\frac{(1-\alpha)^2}{4}$ are positive when $x(t+1) \neq x(t)$. Hence, $h(t) - h(t+1)$ is a sum of positive terms.

5.5 Remark about the continuous-time case

When values change continuously with time, we have that the previous proof cannot be applied. We can still prove that the algorithm is correct using techniques from control theory.

The new proof is based on the concepts of eigenvalues and eigenvectors which are commonly used in control theory to determine whether a linear system is stable and it is similar to the proof presented in [24].

One of the main results in control theory states that:

Theorem 3 *A system whose dynamics can be described by $\frac{dx(t)}{dt} = Ax(t)$ is asymptotically stable if and only if all eigenvalues of the matrix A have strictly negative real part and is unstable if any eigenvalue of A has strictly positive real part.*

This means that a physical system whose dynamics is described using the matrix A changes with time and reaches a configuration where the system remains unchanged forever (i.e. an equilibrium) whatever the initial configuration is if and only if the eigenvalues of the matrix A are negative. From its definition, an asymptotically stable equilibrium point is an eigenvector of the matrix A corresponding to the eigenvalue 0.

Our multi-agent system evolves according to equation 5.2, whose dynamics is specified by

$$\frac{dx(t)}{dt} = -Lx(t) \tag{5.23}$$

where L is the Laplacian matrix. We have that the value 0 is an eigenvalue of the matrix L directly from the definition of the Laplacian matrix of a graph. This is because every row of the matrix sums 0. The corresponding eigenvector is the vector with all coordinate equals to 1, which we denote by u.

We state the Gersgorin disk theorem, which specifies the disk of the complex plane where the eigenvalues of the matrix L are located.

Theorem 4 *Let G be a digraph with Laplacian L with maximum out-degree d. Then all the eigenvalues of L are located in the disk centered at d with radius d in the complex plane.*

Hence, the real-part of the eigenvalues of $-L$ are non-positive. We focus on the eigenvalue $\lambda_0 = 0$ because a linear combination of its eigenvectors is the equilibrium point (i.e. the steady-state) of the system. By definition, the multiplicity of λ_0 is 2; this is because it is given by the difference between the number of rows of L and the rank of L, which is the number of non-zero rows of L. The other eigenvector corresponding to λ_0 is the vector with equidistance values which we denote by v and formally defined as follows:

$$v = \left(0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}\frac{N}{N}\right)^T \tag{5.24}$$

assuming to set $C = 1$.

We can state the main results of the section.

Proposition 5 *Given the system described by the graph G, the protocol in equation 5.2 globally asymptotically solves the equidistance problem. The dynamics of the system asymptotically converges to the equilibrium point x^* given by v.*

We can prove the proposition as follows. By applying theorem 4, we get that all eigenvalues of $-L$ have negative real-parts except for the eigenvalue $\lambda_0 = 0$. Hence, the system which evolves according to equation 5.2 is stable.

The eigenvalue λ_0 has multiplicity 2. Hence, x^* can be obtained as a linear combination of the set of eigenvectors u, v. Given the constraints on the fixed agent positions, we have that v is the only possible candidate. Hence, the system reaches an equilibrium configuration given by vector v.

5.6 Generalized equidistance problem on a square

We can further generalize the equidistance problem to the case when agents are deployed along a square. The goal of the agents is to reach an equidistance configuration with respect to both x and y, i.e a grid configuration.

The problem can be formally stated as follows. The system consists of $(N+1)^2$ agents; each agent is associated to a unique id given by a pair of values (i, j) where $0 \leq i, j \leq N$. Agents with id $(0, 0)$, $(0, N)$, $(N, 0)$ and (N, N) remain fixed at position $(0, C)$, (C, C), $(0, 0)$ and $(C, 0)$. All other agents are able to move around.

The communication protocol changes. We define an agent (i, j) to be a border agent if i or j belong to the set $\{0, N\}$. The communication can be summarized as follows:

- agent (i, j) is stationary. It cannot receive messages.
- agent (i, j) is a non-border agent. It can receive messages only from agents $(i-1, j)$, $(i+1, j)$, $(i, j+1)$ and $(i, j-1)$.

- agent (i, j) is a mobile border agent. When $i = 0$ or $i = N$, it can only communicate with $(i, j-1)$ and $(i, j+1)$. If $j = 0$ or $j = N$, it can only communicate with $(i-1, j)$ and $(i+1, j)$.

The goal of the mobile agents is to reach a grid configuration, i.e. a configuration where agents are equidistant with respect to both the x and the y coordinate; this means that the agent with id (i, j) should be positioned at the point $(\frac{Ci}{N}, \frac{Cj}{N})$.

For the bi-dimensional case, the adjacency matrix A has four zero rows in correspondence of the stationary agents. The remaining rows can have either two (if they are border mobile agents) or four adjacents (otherwise).

An instance of the problem is depicted in figure 5.1. The network consists of 9 agents randomly deployed which interact with the others to reach a grid configuration.

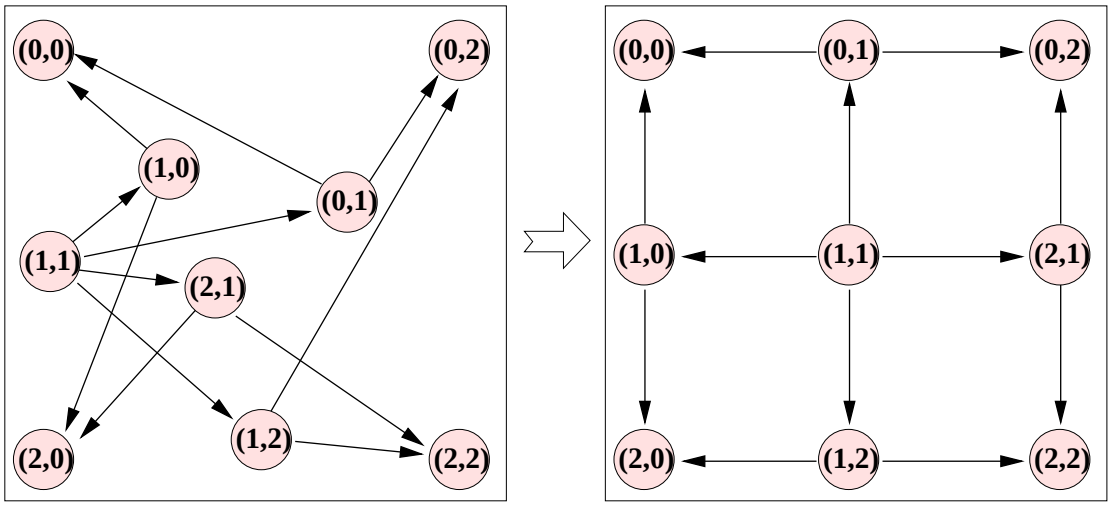


Figure 5.1: An instance of the generalized equidistance problem on a grid.

5.7 A synchronous algorithm for the bi-dimensional case

The dynamics of the system can be described by the same equation given for the one-dimensional case where the Laplacian matrix is obtained from the new adjacency matrix. We report the protocol below:

$$x_i(t+1) = x_i(t) - \gamma \sum_{j \in N_i} (x_i(t) - x_j(t)) \tag{5.25}$$

where the x denotes the position of the agent (i.e. its x and y coordinate). We can apply the same technique to prove that the system converges to the grid configuration.

In the continuous case, the multiplicity of λ_0 becomes 4 and the corresponding eigenvalues are

- The vector with all equal components

$$u = (1, 1, \dots 1)^T \tag{5.26}$$

- The vector which gives the x-coordinate of the agents:

$$w = (u, u, \dots u)^T \tag{5.27}$$

- The third eigenvector corresponds to the y-coordinate of agents

$$z = (v_n, v_n, \dots, v_n, v_{n-1}, v_{n-1}, \dots, v_{n-1}, \dots v_0, v_0, \dots, v_0)^T \tag{5.28}$$

  where v_i is the i-th coordinate of vector v.

- The last vector is

$$s = (ls(0, 1, n), ls(\frac{1}{n-1}, \frac{n-2}{n-1}, n), ls(\frac{2}{n-1}, \frac{n-3}{n-1}, n), \dots, ls(1, 0, n))^T \tag{5.29}$$

  where $ls(x1, x2, n)$ is the compact formula for a vector with n components containing n values equally spaced between the two extremes $x1$ and $x2$.

We get that (u, v) is the equilibrium vector with respect to the constraints on the stationary agents of the new problem.

Chapter 6

Conclusions

We have presented performance analysis and simulation results of algorithms that compute global functions out of local interactions on multi-agent systems in presence of adversarial environments. We have discussed two main agent schemes. In the first scheme, sub-systems behave like a centralized system: they solve their specific optimization sub-problem with a central algorithm. We have given examples where self-similarity produces the optimal solution and other where it fails to do so, and carried out performance analysis on some specific problems. We have evaluated the impact of group size, operation failure and locality on group systems. We have then discussed a synchronous technique, first presented in [24]. Such technique is based on a different idea than self-similarity. The protocol is completely decentralized but synchronous and each agent makes a local update using values of its adjacent. We have investigated its applicability in the context of robot formations.

Much work remains to be done in distributed algorithms applied to multi-agent systems. This includes research in the following directions:

- Termination Detection. In this thesis we have implemented a global termination condition where the computation terminates when simultaneously all agents store some value arbitrary close to their true final value. This is a reasonable choice for a simulation environment; however, in a real scenario, agents do not have enough resources to infer the global state of the network. We would like to investigate algorithms that combine local termination conditions into global conditions and investigate the impact of local termination conditions on the time complexity of the algorithm.

- Asymmetric game theory. The environment has been modeled as a naive adversary of the system. It randomly disables agents and/or communication links. It makes its decision regardless the current state of the system. In a future continuation of the work we would like to model the environment as an active opponent of the system using a game theoretical approach. Furthermore, we would like to investigate the impact of environmental attacks on the convergence of the discussed techniques and design new algorithmic techniques that are able

to take advantage of those attacks.

- Mathematical models for multi-agent systems. We have modeled the system using graphs which is a good model for static systems. In our case, the structure of the system changes continuously; hence, agents should update their knowledge of the graph in an adaptive manner. We would like to investigate mathematical modeling tools which allow to extend the graph model on the basis of the received information.

- Amount of information. In the problem of finding the second smallest value we have pointed out that a naive self-similar algorithm does not work. We have shown that the problem can be solved when agents keep track and exchange both the smallest and the second smallest value of the sub-system. We would like to investigate the minimum amount of information that each agent should store and communicate for solving the problem.

- Message-passing algorithms. The techniques discussed in this thesis can be applied to many practical problems, e.g. robot formations. They give promising results in simulation environments. However, in general, it is very difficult to implement synchronization protocols. We would like to investigate distributed solutions based on asynchronous and decentralized message passing algorithms.

Bibliography

[1] I. Alylidiz, W. Su, Y. Dankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[2] K.J. Astrom and R.M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Manuscript, 2007.

[3] P. Bauer, M. Sichitiu, R. Istepanian, and K. Prematne. The mobile patient: Wireless distributed sensor networks for patient monitoring and care. In *Proceedings of the 2000 IEEE EMBS Interantional Conference on Information Technology Applications in Biomedicine*, pages 17–21, 2000.

[4] V.D. Blondel, J.M. Hendrickx, A. Olshevsky, and J.N. Tsitsiklis. Convergence in multiagent coordination consensus and flocking. In *Proceedings of the Joint forty-fourth IEEE Conference on Decision and Control and European Control Conference*, pages 2996–3000, 2005.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 20–41, 2001.

[6] K.M. Chandy. Properties of concurrent programs. *Formal Aspects of computing*, 6(6):607–619, 1994.

[7] K.M. Chandy and M. Charpentier. Multi-agent collaboration in adversarial environments. In *Submitted to the Twenty-Sixth Symposium on Principles of Distributed Computing (PODC 2007)*, 2007.

[8] K.M. Chandy and B.A. Sanders. Predicate transformers for reasoning about concurrent programs. *Science of Computer Programming*, 24(2):129–148, 1995.

[9] S. Chatterjee and E. Seneta. Towards consensus: some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977.

[10] F. Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–188, 1991.

[11] R. D'Andrea and R. Murray. The roboflag competition. In *Proceedings of the American Control Conference*, pages 650–655, 2003.

[12] M.H. DeGroot. Reaching a consensus. *Journal of American Statistical Association*, 69(345):116–121, 1974.

[13] J.A. Fax and R.M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49:1465–1476, 2004.

[14] G.F. Franklin, J.D. Powell, and A. Emami-Naeni. *Feedback control of dynamic systems*. Addison-Wesley, 2002.

[15] A. Galleni and D. Powell. Consensus and membership in synchronous and asynchronous distributed systems, 1996.

[16] D. Goldschlag. *Mechanically Verifying Concurrent Programs*. PhD thesis, University of Texas at Austin, 1992.

[17] R. Guerraoui and A. Schiper. Genuine atomic multicast in asynchronous distributed systems. *Theoretical Computer Science*, 254(1–2):297–316, 2001.

[18] Y. Hatano and M. Mesbahi. Agreement over random networks. *IEEE Transactions on Automatic Control*, 50(11):1867–1872, 2005.

[19] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor network. In *Proceedings of the sixth International Conference on Mobile Computing and Networking*, pages 56–67, 2000.

[20] A. Jadbabaie, J. Lin, and A.S. Morse. Coordination of groups of mobile autonomuos agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[21] N. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1997.

[22] M. Mehyar, D.P. Spanos, J. Pongsajapan, S.H. Low, and R.M. Murray. Distributed averaging on asynchronous communication networks. In *Proceedings of the forty-fourth IEEE Conference on Decision and Control*, pages 7446–7451, 2005.

[23] C. Nelson and D. Fitzgerald. Sensor fusion fon intelligent alarm analysis. *IEEE Transactions on Aerospace and Electronic Systems*, 12(9):18–24, 1997.

[24] R. Olfati-Saber and R.M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, September 2004.

[25] T. Pham, D. Scherber, and H. Papadopoulos. Distributed source localization algorithm for acoustic ad hoc sensor networks. *Sensor Array Multichannel Signal Process Workshop*, June 2004.

[26] J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the first ACM International Workshop on Wireless Sensor Networks and Application*, pages 99–97, 2002.

[27] W. Ren, R.W. Beard, and E.M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference*, pages 1859–1864, 2005.

[28] A. Rodriguez-Angeles. Cooperative synchronization of robots via estimated state feedback. In *Proceedings of the fourty-second IEEE Conference on Decision and Control*, pages 1514–1519, 2003.

[29] R. Day P. Nag A. Williams W. Zhang S. Clavaski, M. Chaves. Vehicle networks: achieving regular formation. In *Proceedings of the American control Conference*, 2003.

[30] D. Scherber and H. Papadopoulos. Locally constructed algorithms for distributed computations in ad hoc networks. In *Proceedings of the third international symposium on Information Processing on Sensor Networks*, pages 11–19, 2004.

[31] D.S. Scherber and H.C. Papadopoulos. Distributed computation of averages over ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 23(4):776–787, April 2005.

[32] A. Schiper. Dynamic group communication, 2003.

[33] K. Soharabi, J. Gao, V. Ailawadho, and G. Pottie. Protocols for self organization of a wireless sensor network. *IEEE Personal Communication Magazine*, 7(5):16–27, 2000.

[34] D.P. Spanos, R. Olfati-Saber, and R.M. Murray. Dynamic consensus on mobile networks. In *Proceedings of the sixteenth IFAC world congress*, 2005.

[35] A. Tahbaz-Salehi and A. Jadbabaie. Consensus over random networks. *Submitted to IEEE Transactions on Automatic Control*, 2006.

[36] J.N. Tsitsiklis, D.P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stocastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, September 1986.

[37] T. Vicsek, A. Czirok, E. Ben Jacob, I. Cohen, and O. Schochet. Novel type of phase transitions in a system of self-driven particles. *Physical Review Letters*, 75:1226–1229, 1995.

[38] C.W. Wu. Synchronization and convergence of linear dynamics in random directed networks. *IEEE Transactions on Automatic Control*, 51(7):1207–1210, 2006.

[39] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. In *Proceeding of the forty-second IEEE Conference on Decision and Control*, pages 4997–5002, 2003.

[40] L. Xiao, S. Boyd, and S. Lall. A scheme for asynchronous distributed sensor fusion based on average consensus. In *Proceeding of the fourth international conference on Information Processing in Sensor Network*, pages 63–70, 2005.

[41] Y. Xu, H. Qi, and P. Kuruganti. Distributed computing paradigms for collaborative processing in sensor networks. *IEEE Globecom*, pages 3531–3535, 2003.

[42] F. Zhao and L. Guibas. *Wireless sensor networks: an information processing approach*. Morgan Kaufmann Publishers, 2004.

[43] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.